AFIT/GST/OS/85M-4

AN ANALYSIS OF

OPTIMAL AIRFIELD ATTACK PARAMETERS

THESIS

Thomas K. Green — Captain, USAF
David A. Roodhouse — Major, USAF

AFIT/GST/OS/85M-4

Approved for public release; distribution unlimited

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GST/OS/85M-4	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENS	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, OH 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification) See Box 19	

12. PERSONAL AUTHOR(S)
Thomas K. Green, B.S., M.S., Capt, USAF David A. Roodhouse, B.S., M.B.A., Maj, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
MS Thesis	FROM ____ TO ____	1985 March	439

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB. GR.
01	05	
09	04	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Conventional Weapons, Weapon Effectiveness Model, Airbase Damage Assessment, Attack Simulation,

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: AN ANALYSIS OF OPTIMAL AIRFIELD ATTACK PARAMETERS

Thesis Chairman: James R. Coakley, Major, USAF

Approved for public release: IAW AFR 190-17.
LYNN E. WOLAVER 1 May 85
Dean for Research and Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
James R. Coakley, Major, USAF	513-255-3362	AFIT/ENS

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

Abstract

This thesis ~~research effort~~ was directed towards developing flexible, operationally-oriented methodologies to assess the effectiveness of conventional airfield attack. Two methodologies were pursued: computer simulation and response surface methodology.

The computer simulation was based on the Attack Assessment Program (AAP), originally developed by the University of Oklahoma for the Joint Technical Coordinating Group for Munitions Effectiveness and modified by Captain Robert N. Miglin, USAF, in 1984, which ~~The program~~ was modified further to simplify input file generation, translated from FORTRAN V to PASCAL, and implemented on microcomputers. The PASCAL version of AAP consists of four separate programs: three to build the input file and a fourth, called AAPMOD, to accomplish the simulation. A user manual provides complete documentation of the new implementation.

The response surface methodology (RSM) demonstrated how to predict system responses in a simplified airfield attack scenario consisting of two runways. A screening design, a first-order analysis, and a second-order analysis were accomplished. The analysis employed least-squares regression and various statistical tests to fit the response surfaces. The report includes an overview of RSM.

The findings ~~of the analysis~~ indicated that response surface methodology was capable of determining optimum operating conditions, but only for very narrow ranges of the input parameters. The methodology would offer little flexibility to aircrews during mission planning. Further, changes to the target airdrome would dictate reaccomplishment of the response surface fit.

The microcomputer program is recommended for ~~timely, conventional airfield attack analysis~~ at the wing and squadron levels.

AFIT/GST/OS/85M-4

AN ANALYSIS OF
OPTIMAL RUNWAY ATTACK PARAMETERS

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Operations Research

Accession For	
NTIS GRA&I	X
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A/1	

Thomas K. Green, B.S., M.S.

Captain, USAF

David A. Roodhouse, B.S., M.B.A.

Major, USAF

March 1985

Approved for public release; distribution unlimited

Preface

A thesis effort is by no means a small undertaking, and in choosing an area for analysis, we sought a topic that would be of wide interest. Naturally, we wanted the analysis to be interesting to ourselves, and in this vein, our extensive backgrounds in tactical air force fighter and reconnaissance missions helped narrow the search. We wanted to spark the interest of our advisor, Major James R. Coakley, a fighter pilot of long standing. Further, we wanted to research an area of interest to the tactical air force community in general. Airfield attack certainly is a complex problem, constantly discussed by a large number of interested parties, and worthy of analysis effort. Hence, the area of airfield attack seemed perfect.

Simply stated, we could not have produced this thesis without the assistance of a number of people. First, we must acknowledge the fine efforts of Captain Robert N. Miglin, who wrote a thesis last year on the same area of research. His project formed the basis for our follow-on work, and it provided a wealth of information and references for which we are grateful. Second, we are most appreciative of our thesis advisor, Major James R. Coakley, for his instruction, advice, assistance, motivation, and continuing support throughout our tenure at the Air Force Institute of Technology, and especially during our thesis research. Third, we thank Lt Col Palmer W. Smith for introducing us to the area of Response Surface Methodology and greatly assisting us in this difficult methodology. Finally, we would like to express our gratitude to Lt Col Ivy D. Cook, Jr., our thesis reader, for his expert opinions and invaluable suggestions.

It would have been impossible to complete this project without the love, understanding, and support from our families. To them, we wish to say a special word of thanks, and remind them that they, too, are an important part of our nation's security because of their efforts to keep us performing at peak capacity.

THOMAS K. GREEN, Captain, USAF

DAVID A. ROODHOUSE, Major, USAF

Table of Contents

List of Figures

List of Tables

Abstract

This thesis

This research effort was directed towards developing flexible, operationally-oriented method~~ologies~~s to assess the effectiveness of conventional airfield attack. Two methodologies were pursued: computer simulation and response surface methodology. The first method examined -- a

~~The~~ computer simulation was based on the Attack Assessment Program (AAP), ~~originally~~ developed ~~by the~~ University of Oklahoma for the Joint Technical Coordinating Group ~~for~~ Munitions Effectiveness, and modified by Captain Robert N. Miglin, USAF, in 1984. The program was modified ~~further~~ to simplify input file generation, translated ~~from FORTRAN V~~ to PASCAL, and implemented on microcomputers. The PASCAL version of AAP consists of four separate programs: three to build the input file and a fourth, called AAPMOD, to accomplish the simulation. A user manual provides complete documentation of the new implementation. The second method--

~~The~~ response surface methodology (RSM) demonstrated how to predict system responses in a simplified airfield attack scenario consisting of two runways. A screening design, a first-order analysis, and ~~a~~ second-order analysis were accomplished. The analysis employed least-squares regression and various statistical tests to fit the response surfaces. ~~The report includes~~ an overview of RSM is included.

The findings of the analysis indicated that ~~response surface methodology~~ RSM was capable of determining optimum operating conditions, but only for very narrow ranges of the input parameters. The method~~ology~~ would offer little flexibility to aircrews during mission planning. Further, changes to the target airdrome would dictate reaccomplishment of the response surface fit. Keywords: Airbase damage assessment.

The microcomputer program is recommended for timely, conventional airfield attack analysis at the wing and squadron levels.

AN ANALYSIS OF OPTIMAL RUNWAY ATTACK PARAMETERS

I. Introduction

Overview of the Report

This analysis investigates optimal attack parameters for fighter and attack aircraft delivering conventional munitions against airfields. The report is divided into chapters corresponding to the major areas of emphasis. The chapters provide background material before describing more detailed implementations. Chapter I is an introduction to the airfield attack problem. Chapter II considers previous efforts in the area of airfield attack. Chapter III describes the Attack Assessment Program--MODIFIED (AAPMOD), the computer model used in this analysis. Chapter IV describes modifications to AAPMOD and its loading program (AAPIN). Chapter V covers the initial screening experiment conducted to eliminate insignificant variables. Chapter VI provides a primer for Response Surface Methodology (RSM), which combines previously developed mathematical tools to find predicted system responses and optimum operating conditions. Chapter VII describes RSM as applied to this thesis effort. Chapter VIII makes use of the response surface fit and the calculated stationary points to highlight potential applications for the output. Chapter IX considers the sensitivity of the analysis to various parameters and distributions used or assumed. Chapter X concludes with a summary of the study and recommended areas for further research.

Background

Since shortly after its inception, airpower has assumed offensive as well as defensive roles. Airpower offers flexibility and speed that are unmatched by other assets available to ground commanders. Ground commanders receive air support directly in the form of close air support, battlefield air interdiction, and deep interdiction. Only the time frame differs as to when the ground commander will benefit from such support. The effectiveness of direct support missions can be eroded significantly by enemy air activity. Consequently, the ground commander is supported indirectly by friendly actions which might limit the enemy's ability to prosecute an effective air war. Indirect support of this type is called "counterair."

The requirement to conduct counterair operations is widely recognized. Air Force Manual 1-1, Functions and Basic Doctrine of the United States Air Force (13), highlights counterair operations as a primary task of tactical air forces. Counterair is a broad term which can include either defensive or offensive operations. An example of defensive counterair might be friendly surface-to-air missiles or friendly interceptor aircraft. Offensive counterair can cover attacks on airborne aircraft behind enemy lines, attacks on lines of transportation feeding into airdromes, and attacks on the airdromes themselves. This thesis will concentrate on attacks against enemy airdromes.

Attacks against an enemy airdrome can employ various means of achieving the same objective -- that objective being to reduce enemy sortie generation capabilities to minimal levels or to render the airdrome unusable for enemy operations. The means can include

conventional, chemical, or nuclear attack against runway or taxi surfaces, unsheltered aircraft, hardened shelters, command posts, operations centers, quarters/dining facilities, maintenance facilities, fuel storage, and others. In some cases, runways and taxiways are the easiest to see since the other facilities can be camouflaged and hardened more easily. Runway camouflage (also known as "toning-down") is underway in many parts of the world. Nevertheless, large surfaces will be easier to find than small structures, and may be quite extensive.

The multiplicity of taxi and runway surfaces at many airfields makes the attack of these surfaces difficult. Aircraft require certain widths and lengths of surface for takeoff. The width must be at least as wide as the landing gear with some margin for error being highly desirable. A common range of minimum widths for fighter operations is 50-75 feet. The required length is a function of aircraft characteristics and payload. Minimum runway length for fighter operations can vary from as little as 3000 feet up to 5000 feet or more. Distances this short usually require partial weapons loads, partial fuel loads, or both. Takeoff distances are usually longer than landing distances for a given aircraft unless landing at heavy gross weights. Aircraft touchdown points during landing are subject to a higher margin of error compared to takeoff parameters. Either takeoff or landing distance could be the critical variable depending on the specific aircraft and situation. If attacks against runways or taxiways cut the surfaces in sections smaller than the minimum dimensions for takeoff and landing, the airfield has been closed.

An alternate approach would be to attack taxiways and other approaches to the runways; however, this would require closer crater spacing. Maintenance personnel could marshall taxiing aircraft around intervening craters via a meandering path where the same craters might prevent high-speed takeoffs and landings. High-speed aircraft have a restricted turn ability while on the ground. Such aircraft must follow what is essentially a straight line along the surface with only minor deviations permitted. High-speed turns might sheer the landing gear or rupture the tires. Slow-speed taxi over cracked and buckled pavement may be possible where high speeds over the same pavement might cause the tires to rupture. This translates into a much smaller required area for taxiing aircraft. Consequently, taxi surfaces must be handled differently in targeting and assessment of damage. Also, crater repair criteria for taxiways are less stringent since the patches do not have to withstand the shock of aircraft landings.

Since airfield attack is notoriously difficult, requiring many sorties and a great deal of ammunition, ground and air commanders are interested in using the available aircraft sorties as efficiently as possible. Target selection and force composition cannot be tested effectively in peacetime due to the expense of destroying airfield facilities; wartime would be too late. Therefore, the analysis of optimal airfield attack options is a natural problem for computer modeling.

In discussions with Air Force Institute of Technology (AFIT) Class GST-84M, Brigadier General Goodson, Deputy Chief of Staff for Plans, United States Air Forces in Europe (USAFE), emphasized the critical need for estimating survivability and effectiveness tradeoffs (20). This was

reiterated in February of 1984 in a class discussion with Lt Col Moriarity, Chief Analyst, Headquarters United States Air Force (HQ USAF) Studies and Analysis. He presented current models for survivability-effectiveness tradeoffs and sought interested parties at AFIT to continue the work (39).

Many of the current airfield attack models are quite large and require a mainframe computer for execution. This is acceptable for force structure analysis at HQ USAF, but aircraft operators will have very limited mainframe computer support at the front lines. However, it is highly likely that microcomputers will be available. Microcomputer analysis of potential airfield attacks could be extremely useful.

High level interest in survivability-effectiveness tradeoff studies has resulted in a series of AFIT thesis efforts. Various airfield attack models were considered by members of AFIT Class GST-84M. Captain Robert Miglin streamlined the Attack Assessment Program (AAP) and associated computer code into a program called "AAPMOD", which returns expected damage levels and probabilities of cutting runway/taxiway surfaces. Another major effort emphasized aircraft survivability during airfield attacks. Captains Michael J. Foley and Stephen G. Gress produced a continuous simulation model which analyzes aircraft probability of survival in a restricted scenario with a fixed attack heading. They considered target damage levels, but did not tie them directly to the survivability aspects of the problem.

<u>Summary</u>. Offensive counterair is a long-standing tactical air mission, necessary for the indirect support of ground commanders. Airfield attack is a valid offensive counterair mission. Many

counterair targets are easy to hide or harden and are usually well defended. Research in this area is still needed.

Chapter I provides the backdrop for further discussion and research. The next chapter expands this background by considering previous research concerning airfield attacks.

II. Historical Development

Existing Methodologies

This section provides brief descriptions of existing models and methods.

The Joint Munitions Effectiveness Manuals (JMEMs) (28) are the accepted standard for assessing weapons effects and levels of target destruction. These documents consider the threat by decreasing delivery accuracy as the threat increases. No consideration is given to whether the aircraft arrives in a position to deliver ordnance. Range and deflection errors for both weapons delivery and ballistics are assumed independent of one another. For large-scale attacks, JMEMs methodologies are complicated and tedious.

TAC Repeller incorporates aircraft attrition from surface-to-air missiles (SAMs) and anti-aircraft artillery (AAA). Flight paths must be programmed in advance. Aircraft maneuvers are not permitted and target damage assessment is not performed (39).

TAGSEM is an Air Force Systems Command model which outputs target damage in a fashion consistent with the current series of thesis efforts concerning survivability and effectiveness. However, weapons lethality and aircraft survivability must be fed into the model as inputs when these are the desired outputs (39).

In 1976, the Rand Corporation developed the Airbase Damage Assessment Model (AIDA) (16). This model duplicates JMEM figures but does not yield survivability figures. It considers point impact weapons only and requires a large computer to run the program. AIDA offers no advantages over JMEMs, and JMEMs are already well accepted.

Attacking Hardened Air Bases (AHAB) (4:232) is another Rand Corporation model which incorporates decision maker value functions to maximize attack results. Value functions are hard to assess. In addition, AHAB attacks are limited to perpendicular runway cuts with a single weapon type. The model is too restricted for this thesis effort.

RUNW was developed at Supreme Headquarters Allied Powers Europe (SHAPE) in the early 70's to assess runway cuts with a single weapon type. No flexibility is allowed in designing attacks (35:17).

In 1967, USAF Studies and Analysis developed a dynamic programming tactical air warfare model (TAWM) (35:9). Since then, newer and more responsive models have been developed. One of these, the Theater Air Warfare Model 84 (TAWM84), requires quantification of many continuous levels of target damage. Some of the inputs are subjective payoff matrices relating aircraft lost to targets destroyed. Developing the inputs for this large model proves quite complicated.

TAC Avenger (4:84) is an air-to-air combat model which produces results for one-on-one engagements. It incorporates Newton's equations of motion in three dimensions and aerodynamic performance equations. It is useful for comparing alternative aircraft. TAC Avenger is typical of models which provide inputs for more highly aggregated models. TAC Avenger does not consider air-to-ground operations.

TACTICS II is the one-on-one air-to-air model preferred by Battilega (4:153). This model is a very detailed simulation of aircraft maneuvers during aerial combat. Air-to-ground applications are not modeled.

TAC Brawler (4:137) is also an air-to-air model which considers few-on-few engagements.

TAC Contender (4:123) is a theater-level two-sided sortie allocation model. TAC Contender uses game theory in search of optimal sortie allocation considering tons of close air support (CAS) munitions delivered. This model has been criticized for identifying local optima which can be far removed from global optima.

TAC Spartan (4:159) was the working title for the campaign-level model which evolved into TAC Gladiator.

TAC Gladiator (4:129) is a campaign-level model of a Central European Scenario. TAC Gladiator considers each classic role of tactical airpower including offensive counterair. It is designed to analyze force structure problems by employing varying forces at the start of the war. This model is too highly aggregated to use for specific airfield attack missions.

TAC Appraiser (4:131) combines exchange ratios and kill results from TAC Gladiator, adds effectiveness measures and costs, and calculates tactical air force structures and cost/effectiveness tradeoffs.

TAC Warrior (4:136,35:12) approaches theater-level analysis from the bottom-to-top where TAC Gladiator and TAC Appraiser aggregate data before calculations commence. TAC Warrior is very large and requires vast numbers of inputs, including inputs from TAC Avenger, JMEN, LCOM FORMAT, TAC Turner, P001, and SAM models. JMEN captures the essence of JMEM models. LCOM FORMAT generates properly formatted input from the Logistics Composite Model (LCOM) for TAC Turner which is a Monte Carlo model of sortie surge operations. The SAM series models various enemy surface-to-air missiles. P001 models anti-aircraft artillery (AAA) engagements.

TAC Thunder is a follow-on model to TAC Warrior, but is still being written by USAF Studies and Analysis. It will be a large model as well.

The Attack Assessment Program (AAP) (27) is a large Monte-Carlo model which allows for variable attacks against airfields using several munitions types and a variety of weapons delivery patterns. This model is the basis for Miglin's thesis which produced AAPMOD (35). AAP is used extensively at Eglin AFB for new munition development. Exhaustive work has been performed by a variety of interested parties. Whitehead (55) has researched the airfield attack problem thoroughly using AAP and concludes that the probability of cutting runways and the expected number of craters on the runway are conflicting goals. He stated that the highest probabilities of cutting runways are achieved by attacking perpendicular to the primary axis of the surface. This was verified by Douglas (15). However, the number of craters on the runway is optimized by attacking along the axis of the runway. Aircrews are reluctant to use this approach because of the chance of missing the surface entirely on a single pass. If an expected value approach is used, these occasional misses are overshadowed by the large number of craters which can be produced by passes which do hit the runway. Of course, all the impacts could lie on one side of the runway, allowing enemy aircraft to use the other half. Whitehead stated that the best compromise between the conflicting goals is to attack at an angle of 30° relative to the runway axis. This work is exceedingly valuable. However, optimal aimpoints and attack parameters (other than the axis of attack) were not mentioned.

Previous Thesis Efforts

In 1981, Leek and Schmitt (32) addressed aircraft survivability in a thesis encompassing Forward Looking Infrared Radar (FLIR) equipped fighter aircraft. They present a detailed approach to modeling SAM and AAA encounters but did not address effectiveness in the form of target damage.

Anderson and Nenner (1) extended this work in 1982 by expanding the SAM and AAA command and control process. Aircraft damage was not assessed as this model addressed Wild Weasel effectiveness in suppressing enemy threat systems.

Kizer and Neal (29) addressed aircraft survivability versus target damage in a close air support (CAS) scenario. Point damages were assessed based on use of cannon and Maverick missiles.

Pemberton (43) assigns aimpoints for perpendicular runway cuts using set theory to find an "open" cell by means of discrete approximation. This wartime tool is designed for fast execution and addresses only precision, singly-released weapons.

Hachida (22) improved on Pemberton's discrete approximations by eliminating redundancies and improving the search algorithm.

Foley and Gress (18) performed a three-dimensional continuous simulation in which a probability of survival was determined for a fixed attack with a limited threat scenario. The attack was assessed for target damage by means of JMEMs routines. The output is a valuable resource, but the continuous simulation model takes too long to execute. For this reason, the output from Foley and Gress's model may well be the key to more efficient deterministic models.

In 1984, Miglin (35) modified AAP, calling the new program the Attack Assessment Program—MODIFIED (AAPMOD). He also developed a program, called AAPIN, to create the complicated file required to run AAPMOD. AAPMOD can be considered a subset of its parent program, AAP. Both programs model airfield attack with Monte-Carlo iterations of a scenario specified by the analyst in the input file. The database contains the number, type, and location of targets to be attacked; number and type of weapons to be employed per attack pattern; number of submunitions per weapon deployed; extent of damage inflicted upon targets for munition impacts; and attack pass information for each aircraft participating in the mission. AAP's additional capabilities include a larger database and multiple attacks on the target complex with damage repair after each attack wave. AAPMOD was designed for shorter execution times and smaller computers; hence the aforementioned features of AAP were deleted. Output available from the programs includes the expected number of hits per target, expected area damaged, probabilities of cutting runways and taxiways, and the expected number of craters to repair before a cut runway could be re-opened.

Limitations of Existing Methodologies

As mentioned in Chapter I, the objectives of airfield attack are to reduce significantly the enemy sortie generation capabilities or prevent aircraft operations from enemy airdrome surfaces. Existing models do not address enemy sortie generation capability adequately because numerous damage mechanisms are involved. With the exception of JMEMs, air-to-ground models in the previous section are limited to attacks against runway surfaces. While attacks against runways are complicated,

they are easier to model than attacks against other airdrome facilities. JMEMs methodologies are too complicated to calculate commensurate reductions in enemy sortie generation capabilities.

Analysis of airfield attacks is performed frequently throughout all levels of the tactical air forces. The majority of models surveyed in the previous section concentrate on theater-level analysis for force structure planning. The smaller models are restricted to precision guided weapons, perpendicular runway cuts, or very limited attack plans. Existing models were not designed to analyze air-to-ground operations between the two extremes; this void needs to be filled.

The requirement to perform theater-level modeling of airfield attack is well documented. The need for analysis of specific airfield attacks is just as real, but not as well documented. Weapons and tactics offices throughout the tactical air forces are responsible for analysis and improvement of mission planning and execution. The responsibility is shared at wing and squadron levels by these offices and the aircrews themselves. The tactical community constantly seeks better ways to fight the war; the aircrews not only are interested in putting the bombs on target, but surviving the mission as well. The aircrews fuse training, experience, and judgement to accomplish effective mission planning and develop heuristics for less experienced aircrews. Few other tools exist for this extremely important task. Successful airfield attacks balance the two conflicting goals of survivability and effectiveness. For a fixed level of survivability, aircrews want to maximize mission effectiveness. The complexity of airfield attacks renders maximum effectiveness an elusive goal. Quantification of the numerous variables of airfield attack, combined

with tactical experience, could provide a useful mission planning tool. This could take different forms: computer programs, deterministic equations, or quick-reference tables and graphs.

To be useful at squadron and wing levels, a computer program would have to be portable, flexible, and compatible with available hardware. A computer program that potentially could fulfill the requirements of squadron- and wing-level mission planning is the Attack Assessment Program--MODIFIED.

Outside JMEMs, few deterministic means exist to produce mission planning equations, tables, or graphs. However, methodologies exist for analysis of mathematical optima which could lead to deterministic planning tools. One example is response surface methodology (RSM):

> Since most input variables are continuous, quantitative variables, RSM seems a promising approach to optimize the damage resulting from an attack plan (35:94,49:170).

The most likely candidate for further research is a combination of AAPMOD and RSM.

Limitations of AAPMOD

Miglin's objectives were to translate AAP from FORTRAN IV to FORTRAN V and run AAPMOD interactively on a CDC 6600 and a microcomputer. This was a notable effort for one individual; Miglin successfully translated AAP to FORTRAN V and ran it interactively on the CDC 6600. However, the AAPIN input generation program for AAPMOD is awkward because of the multitudinous inputs required at one sitting. These are placed in a separate file which cannot be changed except with detailed knowledge of AAPMOD. Miglin recognized the limitations of AAPIN and recommended several enhancements to ease the user burden of

database generation (35:91-92). One suggestion is that "AAPIN should be prettied to further enhance useability." Experimentation with the latest version of AAPIN revealed a strong need to improve the program: some input prompts are confusing, various data inputs are redundant, and very little ability exists to recover from a mistake during excessively long interactive sessions. Furthermore, minor changes in data require repeating the entire sequence of inputs.

As it existed when Miglin graduated from AFIT, AAPMOD required further verification and validation. The need for further verification of the FORTRAN code became apparent with a review of the compiled listing of program AAPMOD: several undefined variables were identified. Miglin validated AAPMOD using a three factor, two level (2^3 factorial) experiment with the following variables: weapon delivery error, mode of release (singles or paired), and axis of attack (35:76-87). Though the results of the experiment were valid, further testing with other variables and more levels would be in order.

With AAPMOD shown to produce valid results, it may well suit the suggested uses Miglin identifies in his thesis summary:

> AAPMOD can be used by aircrews and tactical planners to optimally employ the conventional weapons they have available to them today. Crews can use AAPMOD to optimally assign weapon systems to targets (35:89).

In order for the program to be of value, the user must be able to understand the design considerations of the input database, correctly build that database, successfully execute the program, and properly interpret the output. In its current form, the user could not begin to understand the structure of AAPMOD without significant effort. Use of AAPIN to build the input file is a vast improvement over the previous

methods of input (text editors and IBM cards), but is still far from the level of friendliness needed for field use.

To generate output useful to aircrews in planning missions, AAPIN must be revamped to provide clear, step-by-step instructions for building the input files. A user manual to accompany the program would be highly useful. Examples and significance of the output must be provided. Furthermore, a description of the unique capabilities of AAPMOD would enhance the end user's ability to tailor attack plans in an efficient manner.

With the versatility of AAPMOD, it might be possible to develop a deterministic quick reference guide for aircrew planning. The guide could take the form of tables and graphs, equations, or separately written computer programs. It might be necessary to develop a combination of the above for each target complex considered as a potential wartime target, or it may be possible to find an even more general, generic set of guidelines.

Finally, more research is warranted for making AAPMOD usable on a microcomputer. Due to the size of the FORTRAN program, AAPMOD will not run on microcomputers with 256 kilobytes (K) of random access memory (RAM). Miglin attributes the large program size to FORTRAN "COMMON" variables that require excessive RAM, and estimates a need for 544K RAM to run AAPMOD. He recommends investigating the feasibility of placing AAPMOD onto microcomputers currently available to the tactical air forces (35:93). Considering the latest government purchases of microcomputers, this goal is in line with available equipment and current technology. Additionally, analysis of classified attack scenarios is made possible with currently available TEMPEST machinery

(for example, the Heathkit/Zenith Z-150 systems and the International Business Machines Personal Computer (IBM PC), both obtainable with RAM in excess of 256K). One way to make AAPMOD compatible with microcomputers containing 256K or less RAM is to translate the FORTRAN V into a different, more compact programming language.

Problem Statement

Air Force planners at all levels require accurate estimates of fighter aircraft survivability versus target destruction (effectiveness). Present methods are either too cumbersome, too time-consuming, or too restrictive to fill the perceived needs at levels ranging from the Chiefs of Staff to operational units. The utility of most existing models decreases drastically when considering specific airfield attack missions.

Implications. Runway attack effectiveness was perceived to be the most lucrative area for immediate study. This thesis concentrates on mission planning at the wing or squadron level for targets assigned by higher headquarters. A complementary thesis effort would be further study of runway attack survivability. The final objective would be to meld these efforts into a single analytical package.

Research Questions

1. Can parameters for airfield attacks be optimized?
2. Can the resulting computer program be reduced to portable form compatible with microcomputers now common in the field?
3. Could this program be used for operational unit mission planning?

Objectives

There are several broad objectives for simulation of airfield attacks. First, the analyst should gain some insight into the problem. This is more likely if an experienced aviator serves as the analyst. Second, potential targets can be tested in a benign simulation environment to determine if they are worth attacking in terms of enemy losses. Third, the differences between various munitions could be explored to find the most effective combinations. Finally, the method and parameters of attack could form the basis for experimentation to determine how best to attack an existing target with the aircraft and munitions on hand. This report and the underlying analysis focus on the last objective: varying parameters to determine how best to attack an enemy airfield.

Specific Research Objectives.

1. Streamline AAPMOD input procedures.
2. Perform verification and validation of AAPMOD.
3. Run a screening experiment using AAPMOD to identify insignificant airfield attack parameters which affect the probability of runway denial.
4. Fit a response surface to AAPMOD using the significant airfield attack parameters mathematically to locate optimal attack parameters.
5. Generate output useful to aircrews in planning missions.
6. Translate the computer programs to run on microcomputers available at wing and squadron levels.

Planned Methodologies

The methodologies employed closely parallel the research objectives. An initial screening design is used to winnow insignificant

factors from numerous airfield attack input parameters. AAPMOD is used extensively during this analysis. Next, response surfaces are fitted to the significant attack parameters to predict the results of an attack, eliminating the need to perform the actual attack. An optimum set of attack parameters is sought using the method of steepest ascent followed by a quadratic surface fit.

Since this thesis effort extends Miglin's work, an expanded discussion of the Attack Assessment Program--MODIFIED follows in the next chapter.

III. AAP/AAPMOD -- Computerized Attack Assessment

Attack Assessment Program

The Attack Assessment Program (AAP) was developed for the Joint Technical Coordinating Group for Munitions Effectiveness under contract F-08635-79-C-0255. Written in FORTRAN IV, AAP is used by the Armament Development Laboratory at Eglin Air Force Base, Florida, as well as 50-60 contractor locations (35:44). This versatile program accepts up to 207 target elements of which 3 can be takeoff and landing (TOL) surfaces (runways) and 43 can be runways or taxiways. The remaining targets are assumed to be structures. Up to 11 target types are identified by their "hardness" codes. Extensive error trapping prevents entering too many targets in a particular category. AAP can store up to 16 different weapons delivery patterns and 6 different weapons. These can be paired in any combination for up to 64 attacks (passes) against selected target elements. Up to 36 weapons can be delivered during each pass; each weapon can have numerous submunitions. Up to 10 separate attacks can be programmed for analysis of airfield closure times versus time to repair damage. This substantial versatility results in massive core memory requirements. AAP cannot run interactively with success, even on the largest of mainframes. It was designed to be run in the batch mode with prearranged input. Entering the numerous bits of deta in an interactive mode would try the patience of even the most dedicated user, who would have to start from scratch each time.

AAP Implementation. AAP uses Monte Carlo iterations to determine the desired output statistics. One iteration represents one entire mission. This mission includes all passes and weapons deliveries for

all aircraft participating in each of the separate attacks. This entire process is repeated for each iteration until reaching a predesignated number of iterations, or sufficient accuracy if that option is selected.

Within an iteration, AAP analyzes one attack at a time. This is intuitively appealing since subsequent attacks come at later times. Success in keeping the airfield closed may depend on previous attacks. Within a particular attack, the individual delivery passes are considered. Random numbers are generated to determine weapon or submunition impact points and probability of functioning. If the weapon functions properly, the resulting crater damage is stored for later analysis. When the iteration has been "flown out," search routines determine the status of target pavements and structures. Within a particular iteration, the runway or taxiway is either cut or not cut. This, essentially, is a step function for the single iteration. Structures are handled slightly differently. The area of the craters which physically infringe on the structure is subtracted from the total area of the the structure. This is reported as the level of damage.

After completing iterations, output statistics are generated. At this point, fractional probabilities of runway cut are computed. Depending on the selected levels of airfield attack parameters and the random number stream for a set of iterations, some iterations will yield runway cuts, and the remainder will leave the runway intact. The fractional probability of runway cut is calculated by dividing the number of iterations in which the runway was cut by the total number of iterations. For instance, if 200 iterations were performed and 150 of the iterations cut the runway of interest, the reported probability of

cut would 0.750. This is the manner in which Monte Carlo sampling returns an aggregate figure of merit from a set of Bernoulli trials.

AAPMOD also calculates the probability of denying TOL operations (probability of cut) for combinations of runways. Since three runways are permitted by AAPMOD, possible combinations are runways 1 & 2, 1 & 3, 2 & 3, and 1 & 2 & 3. At the end of each iteration, the program compares the status of the runways. If all the constituent runways of a combination have been cut, a counter is incremented. When all iterations are complete, the combination counters are divided by the number of iterations to calculate the probability of "cut" for the various combinations. Thus, the combined probabilities of cut are calculated by multiplying the results of Bernoulli trials for the individual runways. However, the final combined probabilities of cut cannot be derived from the product of the individual probabilities of runway cut. Table 1 depicts a 10-iteration example of how probabilities of cut are calculated.

	Runway						
Iteration	1	2	3	1 & 2	1 & 3	2 & 3	1 & 2 & 3
1	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0
3	0	1	0	0	0	0	0
4	0	1	1	0	0	1	0
5	1	0	0	0	0	0	0
6	1	0	1	0	1	0	0
7	1	1	0	1	0	0	0
8	1	1	1	1	1	1	1
9	1	0	1	0	1	0	0
10	1	0	1	0	1	0	0
Pr(cut)	0.6	0.4	0.6	0.2	0.4	0.2	0.1

Table 1. AAPMOD Probability of Cut Sample Calculations

AAP/AAPMOD Output. This section describes the output available from AAP/AAPMOD. The reader should refer to Appendix E during this discussion. AAP output starts with an echo check of the raw input file, if this option is selected. (The echo check is not available in the TURBO PASCAL version.) The raw input file is described in detail at Appendix D. The first values of interest are the confidence limits for the simulation. These limits are the half-interval centered on the expected number of hits for the listed target element. AAP picks the target element with the greatest number of hits for the confidence interval. The target element also can be identified by checking the "EXP NO. HITS" section immediately following the confidence limits. The confidence interval is based on the Student's t-statistic. A separate standard deviation "SIGMA" is reported for the expected number of hits on each target element.

The output continues with the expected area of damage by target element and its standard deviation denoted by "EXP AREA DAM" and "SIGMA." This is suppressed if total damage area calculations are suppressed. Calculation of total damage area increases computer time since overlapping craters must be taken into account. This factor is irrelevant for determining probability of cut. Next, the target group of the target element is listed. Target groups are a matter of convenience in the bookkeeping of the damage. The area of group damage is simply the sum of constituent target element damages. The target group standard deviation is tracked from iteration to iteration and is calculated separately.

It should be noted that the initial lines of output summarizing the expected number of hits and expected area damaged contain the only

output available for structures. Repair of structures is not considered.

The next section of output contains data for TOL surfaces and major taxiways. Probabilities of cut and expected number of craters are listed for each major surface along with their standard deviations. These are labeled "PROB CUT" and "SIGMA." The "PROB CUT" is actually the probability of denying TOL operations since more than one cut may be required for long surfaces. A cut is defined as a series of craters sectioning the surface into areas smaller than the minimum clear length and width required for TOL operations. Next is the expected number of craters which close the minimum clear strip which will be easiest to repair. This amounts to a clear strip which can be opened by repairing the fewest craters, which would certainly be the object of the base engineers at the attacked airdrome. The method of determining where the best strip is located is not important for this analysis. The expected number of craters to repair would be important for repair time estimates and the time-sequenced attacks available in AAP. This value is labeled "EXP NO CRATERS" along with its standard deviation "SIGMA." Suppressed output is denoted by a string of asterisks or "N/A."

Related to the expected number of craters to be filled is the expected area which must be filled, "EXP AREA FILL," and its standard deviation, "SIGMA." Since fractional craters can be filled to open a minimum clear strip, there is no direct correlation between expected number of craters to fill and expected area to fill.

The next portion of output is not used for AAPMOD. AAP has provisions for multiple attacks and crater repairs between successive attacks. The expected number of craters filled between attacks and its

standard deviation are labeled "EXP NO FILLED" and "SIGMA." The subroutines used to calculate number of craters filled and repair times have been retained in AAPMOD, but are not used since only one attack is permitted.

The output concludes with statistics on approaches to a TOL strip. AAP considers either an open minimum clear strip (in which case the attack failed) or the easiest minimum clear strip to repair and then calculates the number of craters that must be filled to reach the clear strip. This is handled in a manner similar to the TOL surfaces although there is no minimum clear length for aircraft taxi and minimum clear width is usually less than for TOL operations.

This data is repeated for all combinations of major TOL surfaces. These combined figures give an indication of probability of cut and difficulty of repair for airdromes with more than one runway.

The last section of output summarizes statistics for minor taxiways. Only the minimum clear taxi width is required for those calculations. The target element and minimum clear width are identified as well as the expected number of cuts, the expected number of craters to fill, and the expected area to fill, along with standard deviations for the latter three.

Attack Assessment Program--MODIFIED (AAPMOD)

As stated earlier, Miglin's aim was twofold. He wanted to translate the original AAP from FORTRAN IV to FORTRAN V and succeeded in this endeavor. The major improvement was the addition of comments and improved logic of IF-THEN-ELSE statements available in FORTRAN V. Miglin's second objective was to run AAPMOD on a microcomputer. In

Item	AAP	AAPMOD
Target Elements	207	112
Taxiways	43	30
TOL Surfaces	3	3
Attacks	10	1
Passes per attack	64	32
Target groups	20	15
Delivery patterns	16	12
Weapons in a pattern	36	12
Hardness codes	11	11
Warhead codes	6	6
Reattacks per aircraft	63	1

Table 2. AAP/AAPMOD Capability Comparison (35:51)

Miglin's case, this was an IBM PC with random access memory (RAM) of 256 kilobytes (256K) and a FORTRAN V compiler. AAP was too large to fit on the IBM PC due to the massive amount of memory consumed by FORTRAN "COMMONs." For those unfamiliar with FORTRAN, COMMON variables are roughly equivalent to global variables.

First, Miglin separated the single large program into a loading program called AAPIN and an execution program called AAPMOD. AAPIN performs some user input error checking and generates an input file for AAPMOD. AAPMOD uses the input file to perform calculations in the same manner as AAP. Miglin significantly reduced the permitted number of target elements, passes, and weapons dropped per pass to reduce storage requirements. He also removed the capability for multiple attacks spaced over periods of time. Table 2 summarizes the capabilities of AAPMOD versus AAP. AAPMOD retains a significant capability for representing potential target areas, weapons, and attack patterns. AAPMOD still handles virtually all realistic attack scenarios. The only

major capability possessed by AAP which is not present in AAPMOD is time sequenced attacks. AAPMOD generates a snapshot in time: results of a single attack are analyzed for the number of craters produced and probability of runway closure. Several subroutines relating to subsequent attacks were retained but are essentially useless without the full facilities of AAP. These subroutines deal with runway repair materials and repair times.

Similar to AAP, the analyst can program up to 112 targets for the airfield to be attacked. Up to 3 of these targets can be runways, up to 30 can be paved surfaces (including taxiways and runways), and the remainder, buildings or structures. Up to six weapons can be programmed. These are represented as a number of submunitions (1 in the case of general purpose bombs and guided munitions) and the associated crater sizes of a submunition against targets of various hardnesses. Next, patterns must be defined. Unlike the aircrews' conception of a pattern, AAPMOD's patterns are the end result of the weapons delivery pass — the pattern of weapon impacts on the ground. Relative to an intended target, these are modified by expected miss distances due to aiming, delivery, and ballistic errors. The aircrew is more likely to consider the maneuvers up to the weapon release point as the pattern, but AAP assumes such maneuvers are accomplished external to the program. Finally, the targets, weapons, and patterns are related in an airfield attack scenario. AAPMOD runs this attack scenario with a user definable number of Monte Carlo iterations.

AAPMOD uses discrete simulation. The events involved for the airfield attack are one-time occurrences. Many of them are stochastic in nature and vary over a continuous range, but do not lend themselves

to continuous simulation. This simulation does not use Lanchester equations or other forms of differential equations characteristic of continuous simulations. The results from numerous Monte Carlo iterations are combined to give probabilities of cut for the runways, expected number of craters, and other statistics. In essence, these statistics are discrete, with decreasing step sizes as the number of iterations increase.

Further Study

Further study of the airfield attack problem is warranted. Although the simplified logic offered by FORTRAN V resulted in faster execution times on mainframe computers, AAPMOD's large size has prevented microcomputer implementation. Furthermore, the loading program needs to be streamlined and equipped with file editing capabilities. Second, aircraft operators will have very limited mainframe computer support at the front lines. However, it is highly likely that microcomputers will be available. AAPMOD could be extremely useful for the assessment of potential airfield attacks. One problem persists in this area: use of large FORTRAN programs on microcomputers. Miglin discovered that AAPMOD was too large of a program to run on a 256K IBM PC. The code segment required 57K, but the data segment required approximately 251K additional memory. Miglin found that the FORTRAN loader would not function properly under these circumstances (35:93). A possible solution would be a translation of the FORTRAN version of AAPMOD into a different computer language compatible with the microcomputer capabilities predominant throughout the tactical air forces.

IV. Specifications of the Analysis

This chapter deals with two major issues. First, the airfield attack problem will be considered in more detail. Particular attention will be directed toward the required inputs for simulation of airfield attacks. Second, the steps taken to model airfield attack will be covered. These steps include some considerations and assumptions for the model and computer coding.

AAPMOD Input Parameters

Banks and Carson (3) emphasize that the real world situation should be the first consideration in simulations. This prevents the misapplication of existing models to loosely related problems. Models are simplified representations of real life. The underlying assumptions may be valid for a particular application, while inappropriate for others. Consequently, this chapter considers real-world considerations first and then examines whether AAPMOD is suitable for optimizing airfield attack parameters.

Miglin (35:26-33) provides a cogent discussion of the parameters of airfield attacks. The parameters are repeated here with a short discussion of each.

Two main categories of parameters will be discussed in this section. Variables which are probabilistic and are not under the control of the aircrew will be referred to as "stochastic" variables. Variables over which the aircrew has direct control will be referred to as "controlled" variables. Some variables may not fit perfectly into a single category. In this case, they will be categorized by their predominant tendencies.

Target Parameters.

Runway Dimensions. The runway dimensions are parameters which define the length and width of a runway. The units may be any the analyst desires, but must be consistent throughout the analysis. This analysis is conducted using feet.

Minimum Clear Dimensions. The minimum clear dimensions are parameters which define the minimum clear width (MCW) and minimum clear length (MCL) for aircraft TOL operations. These will vary among different aircraft. For this analysis, MCL equals 4000 feet and MCW equals 50 feet.

Probability of Reaching the Target.

Navigation Error. Navigation error is a stochastic variable that depends on aircrew proficiency and aircraft systems. Combined with the probability of being shot down by hostile fire, this amounts to the probability of the aircraft arriving in position to deliver ordnance.

Survivability. Survivability is a stochastic variable which quantifies the chances of evading various threats to the mission. Separate probabilities usually are calculated for probability of arriving at the target for the first time versus returning for a reattack. The probability of surviving for a reattack is often the lower of the two since the enemy has been alerted by the first attack. Survivability will be considered fixed for this analysis, but is a primary concern for a follow-on analysis of survivability-effectiveness tradeoffs.

Ability to Engage Target.

Aiming Error. Aiming error is a stochastic variable that depends on aircrew proficiency and aircraft navigation systems. This

variable accounts for the ability to find the preplanned aimpoint on a line feature given that the target complex is in sight. For runways, AAPMOD uses a triangular distribution with zero for the mean and 1000 feet for the end points. This is discussed further in Chapter IX under sensitivity analysis. Aiming error is assumed to be negligible for targets other than runways.

Release Mode. Release mode is a controlled variable which dictates the manner in which weapons are released. This usually includes the release of bombs singly or in pairs. Other modes may be possible depending on the weapon release system.

Release Interval. Release interval is a controlled variable which specifies the time interval between weapons releases. Release interval can be based on time for older aircraft or desired munition ground spacing for newer aircraft with computed release systems. Release interval is usually quite precise unless a system malfunction occurs; malfunctions are very rare.

Number of Pulses. The number of pulses is a controlled variable which is set in the armament system to release the desired number of munitions.

Release Altitude. The release altitude is a controlled variable which, among other factors, will affect the range and reliability of munitions dropped or fired. Errors in release altitude can be reduced through aircrew training or use of an automated release system, but will still have some stochastic characteristics in that nearly every release will have an associated altitude error. This can be compounded by errors in cockpit altimeters.

Release Speed. Release speed is a controlled variable which affects the range of munitions expended. This is subject to the same considerations as release altitude.

Dive Angle. Dive angle is a controlled variable which affects the range and reliability of munitions expended. Dive angle has effects similar to release altitude and release speed.

Axis-of-Attack. The axis-of-attack is a controlled variable which dictates the relative azimuth an attack pass makes with respect to a target. This may be restricted by available navigation aids or enemy defenses. The axis-of-attack will affect the orientation of the weapons on the runway and may be related to deflection errors. Deflection errors are more likely if the axis-of-attack lies perpendicular to a strong crosswind.

Delivery Error. Delivery error is a controlled variable in that aircrews can be trained to improve accuracy. Delivery error is distributed as a bivariate normal corresponding to possible errors in range and deflection. Range errors lie along the aircraft's ground track while deflection errors are referenced perpendicular to the ground track. These separate components can be represented by standard deviations of the distribution, or, more commonly, as range error probable (REP) and deflection error probable (DEP). The concept of REP and DEP is that 50 percent of munitions expended will impact inside the range error probable or the deflection error probable. Note that a munition can impact inside the range error probable and still fall outside the deflection error probable. REP is usually larger than DEP, since it is easier to fly over a target than to release the weapons at just the right point in space. Finally, if only a CEP is known, this is

entered by inputting the CEP value for both the REP and DEP. (When REP and DEP are the same, the bivariate normal distribution degenerates to a circular normal distribution.)

Ballistic Dispersion. Ballistic dispersion, also known as ballistic error, is a stochastic variable which defines the probability distribution of a munition's trajectory. As with delivery errors, ballistic dispersion is separated into range and deflection components. This is handled in the same manner as delivery errors in that the interactive programs deal in REP and DEP while the raw data are stored as standard deviations for input to AAPMOD.

Weapon Pattern. The weapon pattern is the actual location of munition impacts compared to the desired center point of a stick or series of bombs. For AAPMOD, this includes interactions of the release mode, release interval, release speed, altitude, and dive angle. The weapon pattern assumes no aiming, delivery, or ballistic errors. AAPMOD applies aiming and delivery error to the center point of the stick and ballistic dispersion to each individual munition during execution.

Weapon Reliability. Weapon reliability is a controlled variable which states the probability of a weapon functioning. Weapon reliability is controlled in the sense that the most frequent cause of a weapon malfunction is releasing the weapon too low. In this case, the fuze does not have a chance to arm prior to impact. Another factor affecting fuze function is the correct setting of switches in the cockpit: fuze options, release modes, and the master arm switch are examples. Additionally, a weapon may not release or arm due to weapon systems malfunctions. Weapon reliability also can be affected by small impact angles that would cause munitions to ricochet or break apart.

All of the previous factors can be controlled by aircrews to a certain degree: meticulous aircrew preflights will reduce the chances of a weapon release malfunction, proper inflight training will reduce cockpit switch errors, and increased aircrew proficiency will decrease release point errors. Weapon reliability also has a stochastic nature in that there is a finite probability that the fuze will not function internally.

Crater Radius. Crater size, defined by crater radius, normally is a stochastic variable which is a function of munition impact angle, depth of penetration, amount of explosive, and weapon design. Crater size is an input to AAPMOD, so it is considered controlled for this analysis.

System Response. System response, or damage to the airfield, is the dependent variable or set of variables of interest. Potential dependent variables will be discussed in more detail later in this chapter.

The preceding variables and parameters provide the first step toward defining the system to be analyzed. The next step is to coalesce the variables into a scenario for analysis.

Test Scenario

A fundamental task of this analysis is to develop a small scenario to test untried portions of AAPMOD and conduct the analysis itself. Figure 1 depicts the test airfield. Attacks against two runway surfaces were chosen for several reasons. First, two runway surfaces narrow the scope of the analysis and streamline some of the more exotic portions of AAPMOD that demand large allocations of computer central processor

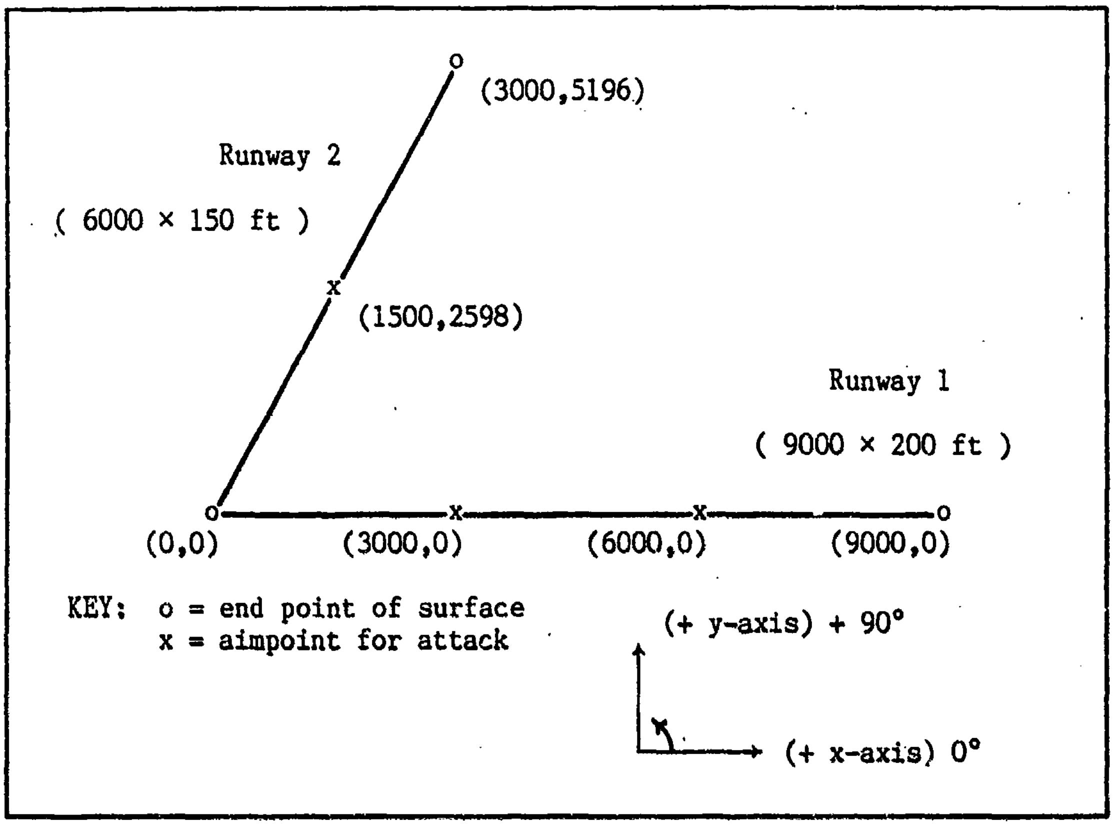

Figure 1. Airfield Layout for the Test Scenario

unit time. Furthermore, a scaled-down program may be the only reasonable option when operating on relatively slow microcomputers. Second, bombing the approaches to a runway usually does not close an airfield for long. Third, the AAPMOD routines that handle structures and repair capabilities are so limited as to render them impractical. Damage to structures is assessed only in terms of crater sizes infringing on the structure. Blast damage and fragmentation damage are not considered. Consequently, structures were also omitted. Provisions are not available for repairing cratered surfaces in AAPMOD. As mentioned in Chapter III, AAP originally provided for a series of attacks spaced in time. The repair module is a throwback to the earlier model for repairs between attacks and was retained only for possible

future use. For these reasons, the analysis concentrates on runway surfaces.

A standard attack will be used against this airfield. Six aircraft will attack the airfield using the same attack heading. The attack heading will be a variable in the analysis, but for a given set of passes, all aircraft will fly the prescribed heading. Three standard aimpoints will be used. These are designed to chop the runway surfaces into 3000 foot sections. Two aircraft will attack each aimpoint. Each aircraft will make only one pass with the same weapon and same attack pattern as the others. This scenario is quite restricted, but will serve as a test vehicle to determine if a response surface fit of this airfield attack is possible.

This thesis is concerned with the interactive effects of attacks against multiple targets such as might be encountered by a mission package assembled for an airfield attack. An alternate approach would have been to consider a single runway requiring only one cut. The reader is encouraged to refer to Peck (42) for a thorough discussion of this approach.

Measures of Merit

There are several potential measures of merit important to the analysis of airfield attacks. Ground and air commanders would be most interested in the time period during which enemy aircraft could not use the airfield after an attack. This could be represented by the number of expected craters to fill or expected area of craters to fill. The expected area to fill would be the more accurate of the two, but requires large amounts of computer processor time to calculate. In

benchmark testing conducted during this analysis, calculations of expected area to fill required up to 20 times the processor time compared to calculations of expected number of craters to fill. In either case, the average times to fill a specified area would be required to estimate airfield closure times.

Equally important is degradation of enemy sortie generation capability; however, this is not available from AAPMOD in its present state.

Another potential measure of merit is the probability of cutting runways or combinations of runways, readily available from AAPMOD. This gives a clear indication of expected success of attack plans. For surfaces requiring more than one cut, or for combinations of surfaces, AAPMOD actually calculates the probability of denying TOL operations. However, this is still reported as the probability of runway cut.

Another possible measure of merit is the number of craters on runway or taxi surfaces. This would give some indication of expected repair times, but the number of craters on a runway does not indicate the presence or absence of a minimum clear strip.

The primary measures of merit for this analysis are the expected number of craters to fill and the probability of runway cut for individual surfaces and combinations of surfaces. Given adequate computer time, expected area to fill replaces expected number of craters as a primary measure of merit.

Structural Model

Different levels of the deterministic and stochastic variables discussed earlier produce various system responses. This analysis

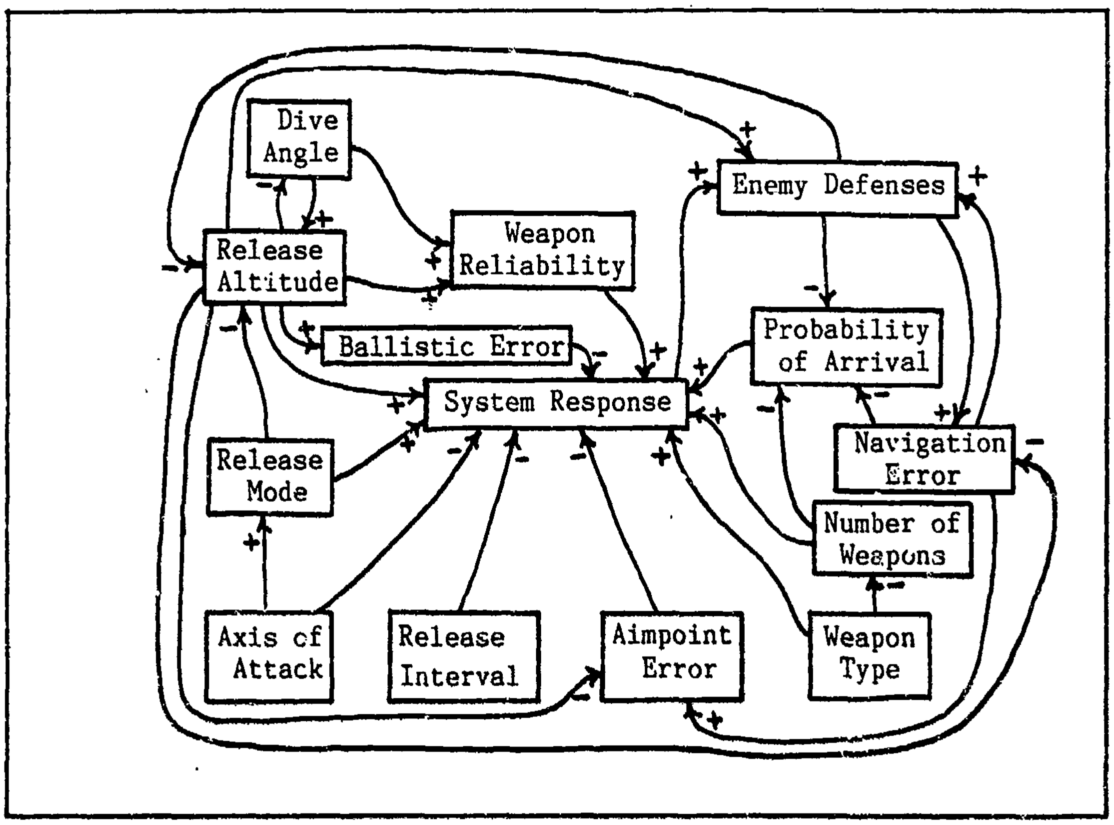

Figure 2. Structural Diagram

considers system response in terms of the measures of merit described in the previous section. For the remainder of the analysis, "system response" includes expected area to fill, expected number of craters, and probabilities of cutting surfaces. The following section considers structural relationships among potential input variables.

Variables Selected for Analysis. There are many potential independent variables. A number of these were discussed earlier in this chapter. This section expands the discussion of input variables for the real-world situation, relates these to AAPMOD and limits the scope of the variables for the analysis. Figure 2 summarizes these variables and their expected influence on the probability of cutting a runway or taxi surface.

To simplify the simulation, the attack is considered from within a reasonable, but unspecified distance of the airfield. This precludes calculating the effect of the probability of engine start on the ground and other unlikely factors. A probability of arriving [Pr(arrival)] at the target area in position to deliver ordnance is the first input and combines all the intervening probabilities up to that point.

The next variables deal with various errors possible in attacking predetermined points of the runway. First, the aircrew may have difficulty in determining the precise location of an aimpoint that is 3000 feet down a 9000-foot runway. AAPMOD uses a triangular distribution for this with the mean at the intended target and the extremities at ± 1000 feet along the runway. This was Miglin's estimate based on the practical experience of a number of fighter aircrews (35:27). Since there definitely is some error in identifying the aimpoint on a line target such as a runway, the triangular distribution was left intact. The triangular distribution is "hard wired" into AAPMOD in routine "TRISUB." It only can be changed by altering the source code and recompiling. Sensitivity of the triangular distribution end points is investigated in Chapter IX.

Next, given an aimpoint intended by the aircrew (not necessarily where intended by the attack planner) there is a possibility of delivering the ordnance on other than the "desired" aimpoint. The second input variables affect the range and deflection due to delivery errors of bomb impact relative to the aircrews' aimpoint. This can be varied from pattern to pattern. There are also range and deflection errors associated with munition ballistics. The standard deviations for ballistic errors were fixed at 30 feet (20.25 feet CEP) for all weapons.

The USAF Fighter Weapons School considers typical ballistic errors to be on the order of 5 milliradians (mils). At 4000 feet slant range between the target and delivering aircraft, 5 mils = 20 feet CEP. Delivery errors are usually 20 mils or more, even with computed release systems. This is intuitively appealing: bombs are fearless in the face of heavy defenses; aircrews are not.

The next inputs are the release point and the release mode of the munitions. AAPMOD allows for sticks of bombs to be dropped singly or in pairs. This corresponds to modern tactical fighter weapons release systems. Some fighters set time spacing between releases while some computed systems permit input of desired ground spacing and release at the appropriate time. The dive angle, true airspeed and bomb trajectories influence where the bombs actually will impact the ground. This makes little difference to AAPMOD as the releases must be translated into intended bomb impacts with the ground. These can be entered manually or with a stick calculation routine built into the input program. The release altitude is an indirect input for AAPMOD; ranging errors and weapon reliability are the direct inputs that relate to release altitude. AAPMOD does not calculate weapon fragmentation damage to the delivering aircraft; however, the probability of an aircraft surviving in the target area to reattack may be input to account for the effect of fragmentation damage. Low releases could prevent the fuze from functioning properly. This can be considered when inputting weapon reliability, although an aggregate figure would be required. This is a simplification, but a reasonable one. The orientation of the stick is set by the attack heading of the aircraft, entered on a pass-by-pass basis.

Once the bombs or submunitions impact the runway, they must form craters to impede aircraft movement. As long as the analysis is restricted to conventional weapons (as is the case with AAPMOD), it really does not really matter what made the hole -- as long as the hole is there. This is exactly the case with AAPMOD. Where the aircrew would think in terms of munitions, AAPMOD uses crater sizes defined for the hardness of the particular target. Different circumstances dictate the crater size a munition will produce. For surfaces, AAPMOD uses two categories of crater sizes, "deny-TOL" and "deny-aircraft-taxi," illustrated by the following example: Assume that the result of a Mk-82 impact prevents aircraft takeoffs (deny-TOL) within a 15-foot crater radius. The same aircraft might be able to taxi slowly over some of the minor rubble and cracks to within a 10-foot radius of the crater (deny-aircraft-taxi). The same munition might result in a different crater size upon impact with a building of equal hardness. Furthermore, it is logical that the near miss of a building might cause some damage, but not as much as a direct hit.

Bombs are discrete in nature, but the damage they render can be considered continuous. Many different permutations of munitions, fuze settings, explosives, and variations in target hardness and impact conditions result in a continuum of crater sizes. Crater size is treated as a continuous variable in this analysis.

AAPMOD uses square craters since they greatly simplify the search procedure for clear takeoff and landing strips. This may appear to be a radical simplification, but the amount of misrepresented area is actually quite small. The side of the square crater is adjusted such that the area inside the square is the same as that of the equivalent

round crater; the side of the square crater is not the same as the diameter of a round crater of equal area. Either half side length or radius can be used to load crater size in AAPMOD. Miglin describes equivalent crater areas in great detail (35:53-54).

The last variables of interest are the minimum clear dimensions for aircraft taxi and aircraft takeoff/landing. As discussed earlier, taxi dimensions will be different from takeoff dimensions.

In summary, there are many types of inputs to AAPMOD. The airfield layout is fixed for the course of the simulation as are minimum clear dimensions for aircraft operations. The location of intended bomb impacts is fixed for a particular pattern, but can be changed from pattern to pattern. Fixed probabilities can be used for aircraft arrival at the target, reattack of the target (if used), and weapons reliability. Aircraft arrival probability can be varied from aircraft to aircraft, while weapon reliability can be changed from pattern to pattern. Runway aimpoint determination is governed by a triangular distribution with fixed parameters. Delivery and ballistic errors are distributed as bivariate normals with provisions to change from pattern to pattern. Obviously, this is a very complicated setup. The reader is encouraged to refer to Miglin (35) for a complementary discussion of input variables.

Computer Coding

Miglin identified the major limitation of the FORTRAN microcomputer version of AAPMOD to be FORTRAN's large memory requirements for "COMMON" variables (35:93). COMMON blocks can be programmed as global variables in other languages with significant memory savings. With this in mind,

an evaluation of available microcomputer programming languages was performed.

The first consideration was given to BASIC, perhaps the most widely used of the microcomputer languages. However, most computers have their own peculiar versions of BASIC depending on the particular capabilities of the hardware, and in some cases, non-programmable read-only-memory (ROM) software. Furthermore, translation of AAPMOD to BASIC would result in an extremely unwieldy program due to the relatively unstructured nature of the BASIC programming language. Even with a successful BASIC implementation of AAPMOD, resulting execution time would be much slower, simply due to the nature of BASIC.

ADA was considered because of its future as the standard programming language of the Department of Defense (DOD). Unfortunately, the ADA compilers available for microcomputers are not a full implementation of the language. Additionally, ADA has such a large program overhead that much of a microcomputer's memory would be usurped by the compiler, leaving little for AAPMOD to use.

MODULA-2, due to its general unavailability and standardization, and COBOL, which is unsuitable for the kind of programming necessary for AAPMOD, were other languages quickly discarded.

Two languages, PASCAL and C, were considered. Both are widely available for microcomputers, reasonably standardized, and very efficient. Either would be suitable for reducing AAPMOD to a size compatible with 256K microcomputers. Both languages are strongly structured and lend themselves to modular programming techniques. They are available on mainframe computers as well, making a new version of

AAPMOD potentially compatible with an excellent variety of hardware. Of the two languages, PASCAL was the language chosen.

Several factors swayed the decision to use PASCAL. PASCAL language constructs are very similar to those in ADA, the new DOD language. A future translation of AAP to ADA would be much easier from PASCAL than from C. A review of current software on the market revealed that of the available C and PASCAL compilers, costs of the PASCAL compilers were generally lower. One particular version of PASCAL, called TURBO PASCAL, was particularly appealing due to its low cost, versatility, and wide range of hardware compatibility. Therefore, TURBO PASCAL was chosen as the language for the microcomputer version of AAPMOD.

TURBO PASCAL is readily available from Borland International, 4113 Scotts Valley Drive, Scotts Valley, CA 95066 at very reasonable cost. Versions are available for the following operating systems: Digital Research's CP/M-80, CP/M-86, and Concurrent CP/M-86; Microsoft's MS-DOS; and International Business Machine's PC-DOS. It is largely compatible with many other versions of PASCAL. TURBO PASCAL compiles very quickly and compactly into command files capable of standing alone and running at relatively quick execution speeds. Since random access memory is a critical resource for AAPMOD, not having to load TURBO PASCAL into memory to run the program is a significant advantage. Another advantage of this compiler is support for the Intel 8087 arithmetic coprocessor unit (for those microcomputers so equipped), resulting in a drastic reduction of time required to run long programs such as AAPMOD.

The next step was to take a critical look at Miglin's version of AAPMOD and AAPIN. AAPMOD was deemed to be reasonable considering what it was designed to do. There are enough comments to make the code

decipherable without adding excessively to the length. The large storage requirements of AAPMOD make it inadvisable to add any more code than absolutely necessary.

AAPIN required some modifications. This program was used to generate the AAPMOD input file. This could be quite an undertaking with up to 112 target elements and 32 passes with 11 different hardness codes, 6 different weapons and 12 different attack patterns. There was no provision to change an existing input file. The analyst would have to run the existing file or spend the time to upload a brand new file for the next run. Alternatively, the analyst could use a computer text editor to change numbers selectively in the input file, but this would bypass all input program error trapping and conceivably result in a latent or run-time error. Finally, AAPIN was largely unstructured, making translation to PASCAL difficult in its current form. The reasonable course of action was to break AAPIN into three smaller programs.

AAPIN has sections dealing with the target array, weapons, attack patterns, and a set of passes planned against the target complex. These sections were ideal dividing points. The current set of programs consists of the following:

AAPTGT - Input or modify target elements.
AAPWPN - Input or modify weapons and/or attack patterns.
AAPMSN - Combine weapons and attack patterns into passes against targets in the target file.
AAPMOD - Execute the file generated by AAPMSN.

This scheme has several advantages. First, target files can be maintained for any number of potential targets. This can be accomplished long before any hostilities erupt and can be updated as

required. The same is true of the possible weapons and attack patterns. The weapon and pattern modules were combined since the individual sections were quite small. Different weapon files could be maintained if there were more than 6 different weapons or if more than 12 patterns were desired. It is easiest to envision this situation if more than one type of aircraft were to be stationed at the same base.

With the predefined target and weapon/pattern databases, the analyst could run AAPMSN to generate a specific mission file. This largely would be a matter of matching targets to attack with weapons and patterns from the predefined database. Most of the time consuming work of entering target, weapon, and pattern data could be performed before a time-critical analysis was required. The file generated by AAPMSN would be used to run AAPMOD. If minor changes were desired, the analyst would have to rerun AAPMSN or use a text editor to change the AAPMOD input file.

Verification and Validation

During the translation process, the FORTRAN V computer code was verified and validated. Verification was limited since the underlying models were not readily available.

Errors were discovered when working with the AAPMOD code. A variable was misspelled in one instance. In FORTRAN, variables need not be declared. The misspelling implicitly declared a single occurrence of a new variable. Five other variables were declared, but not used anywhere in the program. Though corrections were made, no effect was noted in the output from runs conducted before and afterward. One subroutine in AAPMOD is never called. This relates to runway repair

which has been deactivated in AAPMOD. Aside from the errors discovered, the code appeared reasonable. The logic was easy enough to follow although more comments and structured programming would have been helpful.

Some validation had been conducted previously by Miglin. He compared AAPMOD outputs with AAP outputs during initial testing of the new product. He found that some minor deviations occurred between the two versions, but these were limited to less than 1.0 percent of the original value. This easily could be attributed to lack of synchronization in the random number stream between the two versions. What is more, the results appeared to be reasonable. Changes which should have altered the probabilities of runway cut produced changes in the predicted direction. However, the high levels of the probabilities of cut were somewhat surprising. As will be seen in the next chapter, only six aircraft participated in the test case attack. These aircraft achieved probabilities of cut for a runway complex with two runways of 0.700 for one set of parameters. This seems quite high for only six aircraft. The answers only can be considered as relative to one another rather than absolute.

Random Number Generation. An area of concern with the FORTRAN version of AAPMOD was in pseudo-random number generation. The random number generator used in AAPMOD is the intrinsic generator found in FORTRAN V as implemented on the Control Data Corporation 6600, CYBER (45). This generator is a congruential generator using modulus 2^{48}. It has passed several tests for random number generators including the Coveyou-MacPherson test, an autocorrelation test, a pairs-triplets test and others described in Knuth (30). In addition, a statistical

experiment was conducted during the initial stages of this analysis using different random number seeds and found that the random number streams were not significantly non-random. The CYBER random number generator passed all aforementioned tests. The analysis proceeded assuming that random number streams were valid and truly "random."

Random number generation was a concern with the microcomputer version as well. Several tests were performed on the TURBO PASCAL random number generator: a period test of 50 million numbers, several replications of 100-bin and 1024-bin chi-square goodness-of-fit tests, and a random walk test of 500,000 numbers. Results of the chi-square tests may be reviewed at Appendix F. Within an epsilon of 0.0000000001, no repeating pseudo-random numbers occurred with more than 50 million generated between the values of 0 and 1. The random numbers were distributed uniformly (within statistically valid limits) across 1024 bins of approximately 32,000 observations each.

The TURBO PASCAL random number generator appears suitable for the purposes of AAPMOD. One particular drawback, however, is that it cannot be seeded. Therefore, AAPMOD results on microcomputers cannot be duplicated precisely from run to run. However, the variance for Monte Carlo iterations numbering more than 50 was so small that this did not prove to be a major stumbling block. Results from the CDC 6600 and the NEC Information Systems Advanced Personal Computer (NEC APC) agree very closely. Subsequent results can stand alone.

To further validate the TURBO PASCAL version of AAPMOD, results from the FORTRAN and PASCAL versions of AAPMOD with the same number streams were compared. For these tests, the number stream was not

random, but served to validate the code. The results were identical within the accuracy displayed in the outputs.

Distributions. With the exception of the triangular distribution mentioned earlier, the only random variate used by AAPMOD is generated from the normal distribution. The exact inverse method is used, although Miglin suggested changing the generator if operations on microcomputers proved too slow with the 300 or so calls made per Monte Carlo iteration (35:63). Further information on this subject appears in Chapter IX.

Microcomputer Compatibility

For full details of microcomputer system hardware and software requirements, refer to the AAPMOD User Manual at Appendix A, Section III.

The TURBO PASCAL loading programs -- AAPTGT, AAPWPN, and AAPMSN -- can be run on virtually any machine with 64K memory or more. All three programs were tested successfully with TURBO PASCAL Versions 1.0, 2.0, and 2.1. The programs also tested successfully on the following systems: a 64K Kaypro 4 running Digital Research's CP/M-80, Version 2.2; a 56K Electronic Control Technology TT-10 system running CP/M-80, Version 2.2 with ZCPR2; a 64K Integrand Single Board Computer (S-100 Super-Quad) running CP/M-80, Version 2.2; and a 256K NEC APC running Digital Research's CP/M-86, Version 1.107, Concurrent CP/M-86, Version 2.00, and Microsoft's MS-DOS, Version 2.11. Files generated by the programs will be completely compatible among machines, operating systems, and versions of TURBO PASCAL.

AAPMOD requires at least 192K. AAPMOD runs quite nicely on a NEC APC with 256K, running the operating systems mentioned in the previous paragraph for the NEC APC and all previously mentioned versions of TURBO PASCAL.

Contingent on the scenario and the machine, execution time can be quite lengthy. The NEC APC with a floating point (Intel 8087) coprocessor unit ran sample test cases in approximately 3 to 8 minutes. These same runs required about 3 to 6 seconds on the CDC 6600 CYBER. Without the 8087 coprocessor, a 16-bit machine with a 5 MegaHertz 8086 central processor unit will require about 7 times as long to run the same program based on results from a NEC APC.

In summary, the loader programs run efficiently on a great variety of microcomputers, including many 8-bit machines. AAPMOD requires a microcomputer with more capability in terms of memory and execution speed. The floating point arithmetic coprocessor is strongly recommended.

With the means of running the verified and partially validated AAPMOD programs, the analysis continued with statistical experimentation. Chapter V begins the experimentation with documentation of the initial screening design.

V. Initial Experimental Procedure

Focus of the Analysis

This analysis focuses on a wing or squadron weapons office with a microcomputer, the AAPMOD series of programs, and a fragmentary order (frag). A frag tasks the unit to attack a particular target (such as an airfield) with specified munitions and a specified number of aircraft. The task of the weapons officer and the aircrews is to determine the best way to attack the airfield. The two variables under the direct control of the aircrew are attack heading and weapon spacing. Note that this is a completely different problem from that of higher headquarters planners. In the latter case, the number of aircraft and munitions loads would be the variables of interest. The focus of the analysis influences the course of the experimentation and the variables considered.

Overall Experimental Design

The design of the airfield attack analysis achieved a precise estimate of system responses to airfield attacks with minimal cost. The analysis made use of a screening design to be described in this chapter. The objective was to fit a response surface to the sample airfield shown in Figure 1 on page 35.

Response surface fit is accomplished more easily as the number of significant independent variables decreases. The screening design was intended to eliminate insignificant variables.

The next step was to use the method of steepest ascent (first-order) and quadratic (second-order) fit of a response surface to AAPMOD output. The response surface was used to identify optimal attack

parameters consistent with analyst judgement. The indicated mathematical optima were infeasible in some cases, but the boundaries of the feasible regions still indicated optimal parameters subject to realistic constraints.

Due to the complexities of airfield attack, the first step of this analysis was to gain an understanding of the overall problem. This was accomplished by considering the impact of variables expected to be significant. The variables germane to the analysis included variables under aircrew control as well as variables not under aircrew control. Uncontrolled variables affect system responses, and may influence the optimal values of the variables the aircrews can control.

For example, if weapon delivery errors are zero (not directly controlled by the aircrew), the optimal attack heading (controlled by the aircrew) is perpendicular to the runway with weapon spacing (controlled by the aircrew) set to span the runway width. In this case, a perpendicular attack concentrates the munitions on the runway surface with the greatest density and highest probability of cut (15,55). In contrast, the adverse effect of large delivery errors on the probability of cut is reduced by attacking the runway from approximately 30° offset from the runway axis and larger weapon spacing.

Major Assumptions

The experimental designs used in this analysis comply with the three basic principles as set forth by Montgomery (38:2). These principles are replication, randomization, and blocking. Replications are employed to estimate error and provide additional degrees of freedom. Randomization occurs in the sense that random number streams

are used. The CDC CYBER was used for this analysis, which provided the capability of seeding the random number generator and synchronizing the random number streams across replications. Finally, blocking was used to separate the effect of the common random number streams from the effect of the airfield attack input parameters.

Initial Screening Design

Factors and their levels are described in Table 3. The rationale for these levels is as follows. The probability of mission survival in Viet Nam was in excess of 0.99. In the 1973 Arab-Israeli War, the Israelis achieved a probability of survival of 0.97 during the peak loss periods. Even for today's high threat environment, the range 0.99 to 0.92 seems broad enough to bracket possible real-life values.

AAPMOD allows virtually any weapon type since they are defined in terms of crater radii. However, the preponderant runway assault munitions are general purpose bombs. The most common general purpose bombs are 500-pound Mk-82's and 2000-pound Mk-84's. Mk-82's are located in munitions storage areas throughout the U.S. sphere of influence since these are used for aircrew training. In addition to training munitions, vast quantities of Mk-82's are stockpiled in Europe for contingencies in that region.

Crater radius is the next input variable. To keep the analysis unclassified, the crater radii were estimated at first. The estimate was updated by subsequent calculations. For the initial experimentation, crater radii for Mk-84 impacts on runway #1 were set at 40 and 15 feet for deny-TOL and deny-aircraft-taxi values, respectively. For runway #2, the values were 50 and 20 feet. For Mk-82's, the values

Code	Factor	High Level (+)	Low Level (-)
A	Pr(arrival)	0.99	0.92
B	Weapon type	Mk-84	Mk-82
	Equiv crater radius		
	Runway #1	40/15 ft	24/9 ft
	Runway #2	50/20 ft	30/10 ft
C	Del error std dev	300/150 ft	100/50 ft
	REP/DEP	202.5/101.25 ft	67.5/33.75 ft
D	Number weapons	12	6
E	Weapon reliability	0.99	0.59
F	Attack heading	170.0°	140.0°
G	Weapon spacing	100 ft	50 ft

Table 3. Factor Levels for 2-Level Initial Screening Design

were 24 and 9 feet for runway #1, and 30 and 10 feet for runway #2. The difference in values for the two runways is due to the different hardness levels of the pavement surfaces.

Updated crater values were calculated using equations developed in Breuer (6) for the two weapons of interest. The methodology is based on classwork in Systems Engineering 5.62, Terminal Effects of Conventional Weapons. Since cratering is mostly a function of weapon blast, the bare charge equivalent weight is found using the following formula:

$$W_e = C\left[0.6 + \frac{0.4}{1 + 2(M/C)}\right]$$

where

C is the weight of the charge in the encased weapon
M is the weight of metal in the casing
W_e is the bare charge equivalent weight

This equation accounts for the amount of blast energy that is absorbed in accelerating fragments. This bare charge equivalent is an equivalent weight of Tritonal which must be converted to TNT. This is done using the formula:

$$W_{TNT} = (1.13)\ (W_{Tritonal})$$

The crater radius is calculated as follows:

$$r_{crater} = (1 \pm 0.5)\ W_{TNT}^{1/3}$$

The larger value was used as an approximation. Table 4 summarizes the resulting calculations.

Weapon	M	C	W_e	W_{TNT}	r_{crater}
Mk-82	339	192	132	149	7.64 ft
Mk-84	1075	945	687	776	13.78 ft

Table 4. Crater Radii for Mk-82 and Mk-84 Weapons

The crater sizes shown in Table 4 correspond to the deny-aircraft-taxi crater sizes for AAPMOD. Some estimation for crater sizes which would deny takeoff and landing was needed. Lt Stephen Colony (9) with the Rapid Runway Repair Team of the Wright-Patterson section of Prime Base Engineering Emergency Force stated that significant buckling and heaving occur from 10 to 15 feet beyond the physical opening of a crater. The 10-foot figure was used for the Mk-82's and 15 feet for the Mk-84's. This puts the deny-aircraft-TOL crater sizes at 17.64 feet for

Mk-82's and 28.78 feet for Mk-84's. Definitions of deny-TOL and deny-aircraft-taxi values are given on page 41.

The next variable of interest was delivery error. There is general agreement in the tactical fighter community that ranging errors are usually larger than deflection errors. This was corroborated by an informal poll of fighter pilots attending AFIT. Deflection error is estimated to be approximately one half the value of ranging error for a particular delivery. For instance, on a bad day, the typical pilot may fling bombs outside of qualification criteria which is 175 feet for the most likely type of delivery for these bombs. Bombing results from tactical air-to-ground ranges and realistic exercises such as Red Flag indicate that peacetime delivery errors will double during wartime.

A minor problem exists when trying to input range and deflection delivery accuracy. Aircrews think in terms of probable error while AAPMOD requires standard deviations for the underlying bivariate normal distribution. These are related easily by the formula:

$$\text{range standard deviation} = \text{range error probable} / 0.675$$

To avoid confusion, AAPWPN accepts probable error figures and then converts them to standard deviations. Echo checks of data within the program reconvert the standard deviations to probable errors prior to display.

For the initial screening design, a 300-foot standard deviation was considered for the high value and 100 feet for the low value. The high value is representative of manual release systems while the low figure is intended to represent the average pilot flying under hostile fire in the F-16 or A-7 with automated release systems. In terms of range error

probable (REP), the high value is 202.5 feet and the low value is 67.5 feet. The corresponding deflection standard deviations are 150 and 50 feet (equivalent deflection errors probable of 101.25 and 33.75 feet).

Twelve bombs were picked as the maximum load with Mk-82's in mind. No current tactical fighter can carry 12 Mk-84's. This figure was chosen to drive the analysis to reasonably high probabilities of cut. If need be, the high figure for Mk-84's can be considered as several sorties for the sake of comparison. A low figure of six bombs was chosen. Virtually all tactical fighters can carry six Mk-82's. The A-10 aircraft can carry six Mk-84's. These figures served their purpose for the initial screening design.

Weapon reliability for general purpose bombs is very good if the bomb is released at the proper altitude and not too low. For dual fuzing, the 0.99 figure is a standard with newer model M904 and M905 nose and tail fuzes. However, some means were needed for dealing with low release altitudes, weapons release system malfunctions, and incorrect cockpit switch settings. Tactical doctrine dictates that the typical aircrew would tend to remain close to the ground as long as possible to avoid hostile fire. Low-altitude ingress would result in increased numbers of dudded bombs due to low releases in contrast with high-altitude ingress. The 0.59 low-end weapon reliability accounts for low altitude releases, weapons malfunctions, and incorrect cockpit switch settings.

Attack headings were chosen relative to runway #1 depicted in Figure 1. Conventional wisdom says to attack a line feature at 30° off the main axis with a stick of bombs. The stick resolves some of the possible ranging errors while the 30° angle resolves some of the

possible deflection errors. Experienced aircrews do not attack along the primary axis of a runway to avoid missing the runway with all bombs to one side or the other. The angling attack helps prevent this. Suspecting 30° as a near-maximum, axes of attack were chosen to bracket this figure. The screening experiment was run using angles of 40° and 10° relative to the main runway. This becomes 140° and 170° in the airfield reference frame. The 170° was considered the high figure.

The last item is weapon spacing. The 50-foot value is approximately equivalent to the minimum intervalometer setting for typical fighters at attack speeds. Increasing the dive angle is one way to shrink the longitudinal distance between weapon impacts, but dive angles in excess of 20° are required before this is very effective. Such large dive angles would expose aircrews to hostile fire for longer periods of time. For this reason, high dive angles were not considered. The 100-foot value is a high-end value considering crater radii and number of bombs carried. A stick of 12 bombs would cover 1100 feet while a stick of 6 bombs would cover 500 feet. Spacing the bombs further apart might decrease the chances of denying a minimum clear strip for aircraft takeoff. It also should be noted that larger inter-bomb distances would mean longer times spent on final approach waiting for the last bomb to release. This would decrease survivability because of increased exposure to hostile fire. Fortunately, computed release systems are not so limited in this regard.

The preceding paragraph concentrated on "singles" release logic. In other words, bombs are released one at a time. If the weapons pylons are closely spaced, two bombs released simultaneously (paired release) will impact in close proximity to one another. There may be some

tendency for the bombs to spread apart or draw closer together due to ballistic dispersion. While the expected distance between the bombs would be the distance between the weapons pylons, subsequent analysis indicated ballistic dispersion was sufficient to increase probability of cut under certain circumstances. Single release logic was used for this analysis. Paired releases are a subject of sensitivity analysis in Chapter IX.

AAPMOD permits aircraft to reattack; aircraft reattack is a potential input option. However, the vast majority of the fighter community would only expose themselves once to hostile fire in the environment of an enemy airfield. Reattacks are not considered in this analysis. Any additional passes could be added in the form of more aircraft participating in the attack.

Other factors could be considered, but the seven factors found in Table 3 were chosen as the most likely to be significant. All other factors are held constant across the entire experiment. Weapons pylons were spaced 20 feet apart. Standard deviations for ballistic dispersion were fixed at 30 feet (see page 39).

Fractional Factorial Design. Several choices were available for initial screening designs. Fractional factorial designs were chosen. Montgomery mentions Plackett-Burman designs but also states "...these designs are identical to those presented earlier in this section" for seven factor designs (38:346). Consequently, the 2_{IV}^{7-3} and 2_{III}^{7-4} designs described in Montgomery were used. The designs are shown in Tables 5 and 6.

The appeal of fractional factorial designs was the drastic reduction in the number of runs required. However, several assumptions

Generators: I = ABD = ACE = BCF = ABCG

	A^*	B^*	C^*	D^* AB	E^* AC	F^* BC	G^* ABC
(1)	−	−	−	+	+	+	−
a	+	−	−	−	−	+	+
b	−	+	−	−	+	−	+
c	−	−	+	+	−	−	+
ab	+	+	−	+	−	−	−
ac	+	−	+	−	+	−	−
bc	−	+	+	−	−	+	−
abc	+	+	+	+	+	+	+

* main effects which are confounded with second- and higher-order interactions of effects

Table 5. 2_{III}^{7-4} Design and Input Matrix

Generators: I = ABCE = ACDG = BCDF

		A^*	B^*	C^*	D^*	AB	AC	AD	BC	BD	CD	E^* ABC	ABD	G^* ACD	F^* BCD	ABCD
1	(1)	−	−	−	−	+	+	+	+	+	+	−	−	−	−	+
2	a	+	−	−	−	−	−	−	+	+	+	+	+	+	−	−
3	b	−	+	−	−	−	+	+	−	−	+	+	+	−	+	−
4	c	−	−	+	−	+	−	+	−	+	−	+	−	+	+	−
5	d	−	−	−	+	+	+	−	+	−	−	−	+	+	+	−
6	ab	+	+	−	−	+	−	−	−	−	+	−	−	+	+	+
7	ac	+	−	+	−	−	+	−	−	+	−	−	+	−	+	+
8	ad	+	−	−	+	−	−	+	+	−	−	+	−	−	+	+
9	bc	−	+	+	−	−	−	+	+	−	−	−	+	+	−	+
10	bd	−	+	−	+	−	+	−	−	+	−	+	−	+	−	+
11	cd	−	−	+	+	+	−	−	−	−	+	+	+	−	−	+
12	abc	+	+	+	−	+	+	−	+	−	−	+	−	−	−	−
13	abd	+	+	−	+	+	−	+	−	+	−	−	+	−	−	−
14	acd	+	−	+	+	−	+	+	−	−	+	−	−	+	−	−
15	bcd	−	+	+	+	−	−	−	+	+	+	−	−	−	+	−
16	abcd	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+

* main effects which are confounded with third- and higher-order interactions of effects

Table 6. 2_{IV}^{7-3} Design and Input Matrix

had to be met. The population needed to be normally distributed. There is a strong historical basis for assuming that delivery and ballistic errors are normally distributed. In fact, the one random variate used in AAPMOD is normal.

Next, the population error variance was assumed homogeneous. This was tested using Bartlett's test. Population error variance was homogeneous at a level of significance of 0.10 or better in all but one case.

For fractional factorial designs, effects must be additive, a large number of treatments are required, and the design must be balanced with equal levels across factors. In this analysis, the effects were assumed additive, treatments were specified by standard designs, and a balanced design with equal levels across factors was used.

Finally, high-order interactions must be assumed negligible. The experiment revealed virtually all 2-factor interactions to be significant, therefore negating this assumption. In some cases, higher-order interactions also were significant, but these were confounded within aliases to be discussed subsequently.

There are some disadvantages to the fractional factorial design, however. Aliasing does take place, balanced designs are required, computations are more complex, and fractional factorial designs may not be available on some commercial statistical packages.

Before proceeding further, a discussion of design resolutions is in order. Resolution III designs have main effects aliased with 2-factor interactions. All interactions must be assumed negligible. This was not the case for this analysis. The resolution IV design has no main effect aliased with any 2-factor interactions, but 2-factor interactions

are aliased with each other. Resolution V designs have no main effect or 2-factor interactions aliased with any other main effect or 2-factor interaction. Throughout, 3-factor interactions and higher are assumed negligible with the rationale that 3-way interactions are difficult to explain in real-life terms and Montgomery's contention that 3-way interactions are seldom significant in real life (38:326). Therefore, the resolution V design would have been the obvious choice. Unfortunately, each time the resolution increases, the number of required design points doubles. The resolution IV design was a good compromise. In addition, the resolution III design was also used to identify significant interactions within the aliases.

Separate design points were run for the resolution III and resolution IV designs. This would have helped to separate interactions within the aliases if enough interactions had been insignificant. The resolution III design depicted in Table 5 and the resolution IV design in Table 6 were run using the design matrices located at Appendix H.

Sample Size and Reliability

The Monte Carlo simulation performed by AAPMOD consists of a user-specified number of replications, called iterations or samples in AAPMOD documentation (27). If the user specifies 200 or less samples, the single run of AAPMOD consists of the specified number of samples. If more than 200 samples are specified and the option is selected, AAPMOD will stop at 200 samples, calculate the number of additional samples required to meet a user-specified level of significance, and complete the calculated number of samples. For a balanced design, 200 iterations were used throughout the analysis.

Three separate AAPMOD replications were accomplished at each design point to test the random number generator and provide degrees of freedom for the analysis. Three seeds drawn from the random number table in Montgomery (38:532) were used for each treatment. Consequently, each treatment was run a total of 600 times: 200 in each of 3 replications.

For AAPMOD runs consisting of 200 iterations, the sample variance of differences between probabilities of runway cut from seed to seed was only 0.030 with mean values of approximately 0.5. The random number streams were not statistically non-random. As a point of interest, the implied number of replications indicated by the sample variance of 0.030 was 1.3 to detect a 0.1 measurable difference in probability of cut between treatments. Using three replications drops the predicted measurable difference to 0.06. It should be noted that future runs could use the AAPMOD facility of error checking to achieve the desired error. This was inappropriate for this analysis because of the balanced designs used; if AAPMOD calculated its termination points, the number of samples in the experiment would have varied among design points, causing the design to become unbalanced.

Initial Experimentation

Problem Analysis. As mentioned earlier, one of the prime objectives in this analysis was to use AAPMOD and evaluate its performance. This involved forming the design matrices located in Tables 5 and 6 and the input matrices located at Appendix H.

The measures of merit for the experiment were probability of cutting runways and expected number of craters (see page 36). Based on their significance, the hierarchy of AAPMOD input variables was the same

Factor	F	p
Pr(arrival)	32	0.0
Weapon type	102	0.0
Delivery error std dev	3477	0.0
Number of weapons	392	0.0
Weapon reliability	249	0.0
Attack heading	19	0.0
Weapon spacing	39	0.0

Table 7. Resolution IV Screening Design Results
10 Runs per Design Point

in every case for both measures of merit. For this reason, only analysis based on probabilities of cut is shown in Tables 7 through 13.

The screening experiment was accomplished in two ways. In one case, the 16 design points of the resolution IV design were run 10 times each, with 20 AAPMOD samples specified per run. In the other case, the same 16 design points were run 3 times each, with 200 AAPMOD samples specified per run. For the resolution III design, only the latter case was used. The object of both approaches was to reduce the variance of the output by trading off the number of runs versus the length of the runs.

Table 7 summarizes the results for the resolution IV design using 10 runs per design point. Only the main effects are shown. The F- and p-values for each AAPMOD input variable are shown. With a level of significance of 0.05, all main effects are significant. In order for a factor to be insignificant, its p-value must exceed the desired level of significance.

Next, a total of 66 runs was performed at 200 iterations each. The reduction from 72 indicated runs to the 66 runs accomplished was a

result of overlapping treatments between the resolution III and resolution IV designs.

A computer program for multivariate analysis of variance (MANOVA) was used to analyze the output of the 66 runs comprising the experiment. Results of the MANOVA's are shown in Tables 8 through 13.

Tables 8 through 10 summarize results for the resolution III experiment. Tables 11 through 13 present the results for the resolution IV experiment.

Tables 8 through 13 provide information concerning the significance of the analysis and the AAPMOD input variables. The first section indicates a level of significance for the regression by means of a p-value. Next, an R^2 figure shows to what extent the input data have been explained by a linear regression fit. An R^2 value of 1.0 indicates that all the data have been explained by the regression. As more variables are added to an experiment, the R^2 value would be expected to increase asymptotically toward 1.0. Following this, Bartlett's test for homogeneity of variance provides a level of significance indicated by a p-value. The last section tabulates regression coefficients and p-values for AAPMOD input variables. The regression coefficients form the linear equation relating input values to a predicted probability of cut. Degrees of freedom are indicated by "d.f." The sums of squares are squared deviations from the mean for a particular variable. Mean squares are the sums of squares divided by degrees of freedom.

ANALYSIS OF VARIANCE

Source	Sum of Squares	d.f.	Mean Squares	F	p
Regression	2.61345	7	0.37335	1152.46	0.0
Error	0.00518	16	0.00032		
Total	2.61863	23			

$R^2 = 0.9980$

Bartlett's test for homogeneity of variance:

$X_0^2 = 16.4298$ d.f. = 7 p = 0.0215

SIGNIFICANCE OF FACTORS

Factor	Regression Coefficient	Mean Square	F	p
Mean	0.4408			
Pr(arrival)	0.0183	0.00807	24.900	0.0001
Weapon type	0.1400	0.47040	1452.030	0.0
Delivery error	-0.2508	1.51002	4661.110	0.0
# weapons	0.1033	0.25627	791.110	0.0
Wpn reliability	0.1058	0.26882	829.782	0.0
Attack heading	-0.0642	0.09882	305.027	0.0
Weapon spacing	-0.0067	0.00107	3.293	0.0884

Table 8. Runway 1, 2_{III}^{7-4} Design

ANALYSIS OF VARIANCE

Source	Sum of Squares	d.f.	Mean Squares	F	p
Regression	1.76335	7	0.25191	345.95	0.0
Error	0.01165	16	0.00073		
Total	1.77500	23			

$R^2 = 0.9934$

Bartlett's test for homogeneity of variance:

$X_o^2 = 13.2419$ d.f. = 7 p = 0.0664

SIGNIFICANCE OF FACTORS

Factor	Regression Coefficient	Mean Square	F	p
Mean	0.6846			
Pr(arrival)	0.0258	0.01602	21.996	0.0002
Weapon type	0.1275	0.39015	535.808	0.0
Delivery error	-0.1354	0.44010	604.412	0.0
# weapons	0.1163	0.32434	445.425	0.0
Wpn reliability	0.1567	0.58907	808.989	0.0
Attack heading	-0.0067	0.00107	1.465	0.2437
Weapon spacing	-0.0104	0.00260	3.576	0.0769

Table 9. Runway 2, 2_{III}^{7-4} Design

ANALYSIS OF VARIANCE

Source	Sum of Squares	d.f.	Mean Squares	F	p
Regression	2.80636	7	0.40091	818.88	0.0
Error	0.00783	16	0.00049		
Total	2.81419	23			

$R^2 = 0.9972$

Bartlett's test for homogeneity of variance:

$X_o^2 = 13.6394$ d.f. = 7 p = 0.0580

SIGNIFICANCE OF FACTORS

Factor	Regression Coefficient	Mean Square	F	p
Mean	0.3831			
Pr(arrival)	-0.0035	0.00030	0.615	0.4444
Weapon type	0.1640	0.64518	1317.810	0.0
Delivery error	-0.2369	1.34663	2750.580	0.0
# weapons	0.1240	0.36878	753.247	0.0
Wpn reliability	0.1206	0.34921	713.281	0.0
Attack heading	-0.0610	0.08943	182.658	0.0000
Weapon spacing	-0.0169	0.00683	13.960	0.0018

Table 10. Combination, 2_{III}^{7-4} Design

ANALYSIS OF VARIANCE

Source	Sum of Squares	d.f.	Mean Squares	F	p
Regression	4.22925	15	0.28195	417.71	0.0
Error	0.02160	32	0.00067		
Total	4.25085	47			

$R^2 = 0.9949$

Bartlett's test for homogeneity of variance:

$X_o^2 = 23.9723$ d.f. = 15 p = 0.0656

SIGNIFICANCE OF FACTORS

Factor	Regression Coefficient	Mean Square	F	p
Mean	0.4378			
Pr(arrival) (A)	0.0378	0.06863	101.674	0.0
Weapon type (B)	0.0449	0.09675	143.335	0.0
Delivery error (C)	-0.2303	2.54610	3772.030	0.0
# weapons (D)	0.1351	0.87615	1298.010	0.0
(AB)	0.0186	0.01669	24.723	0.0
(AC)	-0.0266	0.03387	50.174	0.0000
(AD)	-0.0324	0.05038	74.630	0.0
(BC)	-0.0041	0.00079	1.174	0.2868
(BD)	0.0030	0.00044	0.649	0.4264
(CD)	-0.0289	0.03996	59.205	0.0
Wpn reliability (E)	0.0759	0.27679	410.064	0.0
(ABD)	0.0576	0.15928	235.965	0.0
Weapon spacing (G)	0.0282	0.03825	56.668	0.0
Attack heading (F)	-0.0209	0.02104	31.174	0.0000
(ABCD)	-0.0093	0.00413	6.112	0.0189

( ) indicates factor code or interaction term

Table 11. Runway 1, 2_{IV}^{7-3} Design

A N A L Y S I S   O F   V A R I A N C E

Source	Sum of Squares	d.f.	Mean Squares	F	p
Regression	2.60465	15	0.17364	261.28	0.0
Error	0.02127	32	0.00066		
Total	2.62592	47			

$R^2 = 0.9919$

Bartlett's test for homogeneity of variance:

$X_o^2 = 19.8483$ d.f. = 15   p = 0.1778

S I G N I F I C A N C E   O F   F A C T O R S

Factor	Regression Coefficient	Mean Square	F	p
Mean	0.6858			
Pr(arrival) (A)	0.0179	0.01541	23.185	0.0000
Weapon type (B)	0.0952	0.43510	654.701	0.0
Delivery error (C)	-0.1056	0.53552	805.799	0.0
# weapons (D)	0.0742	0.26403	397.293	0.0
(AB)	0.0048	0.00110	1.658	0.2071
(AC)	-0.0464	0.10360	155.891	0.0
(AD)	0.0371	0.06601	99.323	0.0
(BC)	0.0017	0.00013	0.201	0.6572
(BD)	0.0348	0.05810	87.427	0.0
(CD)	0.0485	0.11310	170.186	0.0
Wpn reliability (E)	0.1358	0.88563	1332.620	0.0
(ABD)	0.0044	0.00092	1.382	0.2484
Weapon spacing (G)	-0.0448	0.09630	144.906	0.0
Attack heading (F)	0.0375	0.00068	1.057	0.3211
(ABCD)	0.0246	0.02901	43.6491	0.0000

( ) indicates factor code or interaction term

Table 12. Runway 2, 2_{IV}^{7-3} Design

A N A L Y S I S   O F   V A R I A N C E

Source	Sum of Squares	d.f.	Mean Squares	F	p
Regression	3.88395	15	0.25893	302.40	0.0
Error	0.02740	32	0.00086		
Total	3.91135	47			

$R^2 = 0.9930$

Bartlett's test for homogeneity of variance:

$X_o^2 = 27.8880$ d.f. = 15   p = 0.0223

S I G N I F I C A N C E   O F   F A C T O R S

Factor	Regression Coefficient	Mean Square	F	p
Mean	0.3452			
Pr(arrival) (A)	0.0519	0.12917	150.855	0.0
Weapon type (B)	0.0710	0.24225	282.924	0.0
Delivery error (C)	-0.1946	1.81741	2122.530	0.0
# weapons (D)	0.1246	0.74501	870.088	0.0
(AB)	-0.0181	0.01577	18.416	0.0002
(AC)	-0.0396	0.07521	87.835	0.0
(AD)	0.0125	0.00750	8.759	0.0058
(BC)	-0.0288	0.03968	46.336	0.0000
(BD)	0.0375	0.06750	78.833	0.0
(CD)	-0.0252	0.03050	35.623	0.0000
Wpn reliability (E)	0.1088	0.56767	662.982	0.0
(ABD)	0.0429	0.08841	103.251	0.0
Weapon spacing (G)	-0.0090	0.00385	4.500	0.0418
Attack heading (F)	-0.0265	0.03360	39.244	0.0000
(ABCD)	0.0206	0.02042	23.847	0.0000

( ) indicates factor code or interaction term

Table 13. Combination, 2_{IV}^{7-3} Design

Most of the factors were significant throughout the analysis. The R^2 value was in excess of 0.99 in every case. Virtually all the variation in the results was explained by the linear regression equation.

For the resolution III analysis of runway #1, only weapon spacing was insignificant at a 0.05 level. Since the p-value was 0.088, weapon spacing would have been significant at a 0.10 level. The remaining p-values were such that the probability of arrival at the target, weapons type, delivery error, number of weapons, weapon reliability, and attack heading were all significant at any reasonable level of significance. For runway #2, the same was true except that attack heading was insignificant. This was to be expected, since the angles from which the runway was attacked were 80° and 110° relative to the runway. This was almost symmetrical about the 90° axis, so the result was reasonable. For the combined case, the only insignificant factor was probability of reaching the target. This anomaly probably was caused by the small interval chosen between the high and low values of this effect as well as confounding interaction terms. Common sense tells the analyst that the probability of getting to the target will affect the probability of cuts. If the true range of probabilities of reaching the target is small, then this factor probably should be discarded as insignificant.

For the resolution IV design for runway #1, only the BC and BD interactions were insignificant. This was reasonable, since the weapons type should not interact with the delivery error or the number of weapons. All other effects and interactions were significant to at least the 0.05 level. For runway #2, the AB, BC, and ABD interactions

and attack heading were insignificant. The probability of arrival should not interact with the weapons load (at least in this model); the weapons type and delivery error should not interact; and the 3-factor interaction involving probability of arrival, weapons type, and number of weapons should not be significant. The attack heading against runway #2 was significant. The high and low values for attack heading resulted in 70° and 80° angles with respect to runway #2. The 10° made enough of a difference to show up with the additional runs and less confounding. Finally, all main effects and tested interaction terms were significant at a level of 0.05 for the combined probability of cut. This included aliases which contained only interaction terms.

Bartlett's statistic is significant at the 0.05 level in several cases. The variance across cells is not completely homogeneous, but the balanced design should be sufficiently robust to handle this small deviation.

An alternate approach is to inspect the regression coefficients themselves. Since they are scaled in a sense, the size and sign of the coefficients may give a clue as to how the system responds. The coefficients are representative for the resolution IV experiment for runways #1 and #2 combined. The probability of arrival has a moderate positive influence on the probability of cut (P_{cut}). The weapon type has a slightly larger positive influence. Delivery error has the largest absolute value coefficient of any. Increases in delivery error decrease the P_{cut}. The number of weapons has a relatively strong positive influence on P_{cut} as does weapon reliability. The latter is due, in part, to the wide range between high and low levels. Weapon spacing has a very weak and inconsistent effect. The attack heading is

not much stronger based on the regression coefficient alone. If the regression coefficients were scaled equally, the coefficients would indicate that delivery error, number of weapons, and weapons reliability should be examined first. Attack heading and weapon spacing probably can be discarded. Probability of arrival and weapon type seem important on an intuitive level and have moderately high regression coefficients. This is supported by analysis of the structural diagram depicted earlier in Figure 2.

Tables 14 and 15 depict the aliases for the screening experiments. Normally, some of the main effects and most of the interactions would be insignificant. Different combinations of insignificant effects and interactions would be pieced together to break out the confounding of the aliases. This is not possible for this analysis, since most main effects and interactions were significant.

Implications. The initial screening experiments achieved the objectives. Valuable experience with the new implementation of AAPMOD and insight into its capabilities were obtained. The random number generators were successfully tested. Finally, all of the AAPMOD input variables chosen for analysis proved to be significant.

The completed initial screening experiments have laid the foundation for further analysis using response surface methodology. However, before applying the tools of RSM to the airfield attack scenario, Chapter VI provides an introduction to response surface methodology.

Generators: I = ABD = ACE = BCF = ABCG

Effect	Aliases *									
	ABD	ACE	BCF	ABCG	BCDE	ACDF	DCG	ABEF	BEG	AFG
A	BD	CE	—	BCG	—	CDF	—	BEF	—	FG
B	AD	—	CF	ACG	CDE	—	—	AEF	EG	—
C	—	AE	BF	ABG	BDE	ADF	DG	—	—	—
D	AB	—	—	—	BCE	ACF	CG	—	—	—
E	—	AC	—	—	BCD	—	—	ABF	BG	—
F	—	—	BC	—	—	ACD	—	ABE	—	AG
G	—	—	—	ABC	—	—	CD	—	BE	AF

Effect	Aliases *				
	DEF	ADEG	BDFG	CEFG	ABCDEFG
A	—	DEG	—	—	—
B	—	—	DFG	—	—
C	—	—	—	EFG	—
D	EF	AEG	BFG	—	—
E	DF	ADG	—	CFG	—
F	DE	—	BDG	CEG	—
G	—	ADE	BDF	CEF	—

* 4-way interactions and higher are not considered, denoted by —

Table 14. 2_{III}^{7-4} Generators and Aliases

Generators: I = ABCE = ACDG = BCDF

Effect	Aliases *						
	ABCE	ACDG	BCDF	BDEG	ADEF	ABFG	CEFG
A	BCE	CDG	——	——	DEF	BFG	——
B	ACE	——	CDF	DEG	——	AFG	——
C	ABE	ADG	BDF	——	——	——	EFG
D	——	ACG	BCF	BEG	AEF	——	——
E	ABC	——	——	BDG	ADF	——	CFG
F	——	——	BCD	——	ADE	ABG	CEG
G	——	ACD	——	BDE	——	ABF	CEF
AB	CE	——	——	——	——	FG	——
AC	BE	DG	——	——	——	——	——
AD	——	CG	——	——	EF	——	——
BC	——	——	DF	——	——	——	——
BD	——	——	CF	EG	——	——	——
CD	——	AG	BF	——	——	——	——
ABD	CDE	BCG	ACF	AEG	BEF	DFG	——
ABCD	DE	BG	AF	——	——	——	——

* 4-way interactions and higher are not considered, denoted by ——

Table 15. 2_{IV}^{7-3} Generators and Aliases

VI. Response Surface Methodology Overview

Basic Definitions

Response surface methodology (RSM) is a collection of mathematical and statistical tools for discovering optimum conditions. This applies to both deterministic and probabilistic problems. This chapter will concentrate on probabilistic problems.

Box and Wilson (5) first proposed RSM in 1951. It should be noted that the techniques involved existed long before that date. RSM initially was intended for finding optimal conditions in industrial settings. Most of the example problems found during research for this chapter involved chemical processes during manufacture. However, RSM is valid for any circumstance in which dependent and independent variables are quantifiable, continuous and related by some sort of "response" function. Extensions might be made to discrete variables but the optimal point may elude the analyst. This is similar to the case where linear programming is applied to an integer programming problem. The results may be unpredictable.

The response function usually involves one dependent variable of interest (y) and one or more independent variables (x's). This takes the following form:

$$y = F(x_1, x_2, \ldots, x_k)$$

where k is the number of independent variables.

The response surface with two independent variables might appear as shown in Figure 3.

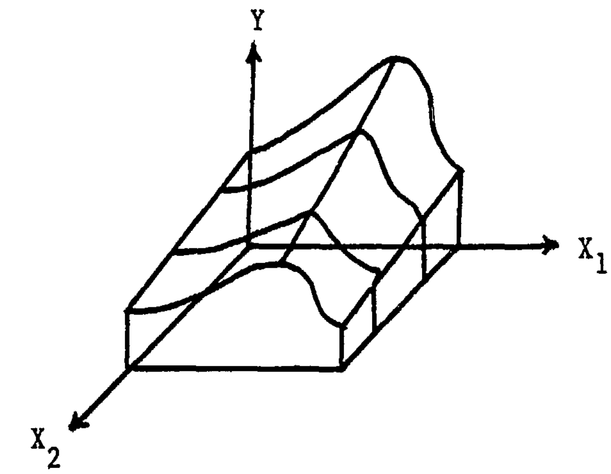

Figure 3. Typical Response Surface

The function F can take many forms but is usually assumed to be a polynomial. This is particularly handy since solutions can be found with relative ease. However, caution must be used when extrapolating results. Polynomial fits are only valid in the immediate area of experimentation. RSM does extrapolate outside the immediate area of experimentation but validates this by comparing experimental responses with predicted responses.

The polynomials could be of any order, but usually are restricted to first-order for initial work and second-order when approaching the optimal point (also called "near stationary" conditions since the optimal point is also a stationary point). The first-order polynomial takes the form:

$$y = b_0 + b_1x_1 + \dots + b_kx_k$$

where the b_i are regression coefficients for $i = 0, 1, \dots, k$.

Including interaction and squared terms, the second-order polynomial with three independent variables appears as follows:

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + b_{11}x_1^2 + b_{22}x_2^2 + b_{33}x_3^2 + b_{12}x_1x_2 + b_{13}x_1x_3 + b_{23}x_2x_3$$

As might be surmised, higher-order polynomials become complicated; RSM usually is restricted to second-order. In most cases, this is a good approximation. Taylor series expansions provide the basis for truncating the series above the second-order terms. For RSM, the analyst should experiment in the immediate vicinity of the newly found optimum point anyway. The more restricted the area, the better is the approximation of a second-order polynomial compared to reality.

Second-order polynomials can accommodate any number of independent variables. However, various interaction terms greatly complicate the problem. This can be alleviated by dropping statistically insignificant variables.

Several steps are required in fitting the desired form of polynomial to the simulation or industrial situation in question. The first step is to identify an area of interest. This could be the status quo or the suspected optimum mix of input variables based on the analyst's judgement. Some knowledge of the actual system is helpful. Experiments or simulations are run in the area of interest. The resulting y-values (system responses) and the x-values (input variables) are analyzed by least-squares regression to find the polynomial fit of the system responses.

First-Order Approximation

If the analyst has limited knowledge of the system, the suspected center point of the problem is assumed to be remote from the actual optimum. A first-order polynomial is assumed applicable in that the slope of a response surface is likely to be fairly smooth until approaching the optimum point. The polynomial is found, fitting a plane tangent to the surface at the present center point. A plane (or hyperplane if more than 2 independent variables) is characteristic of the first-order approximation. The math remains the same with hyperplanes, but conceptualization becomes more difficult.

The method of "steepest ascent" is accepted as the best approach to the optimum point. The analysis moves in proportion to the regression coefficients or "climbs the hill" perpendicular to the contour lines. The main question is how far to step between iterations; the analyst must exercise judgement to choose the appropriate step size. Backtracking with smaller steps is always an option.

Eventually, the optimum point will be approached, at which time the slope will level out. The second-order polynomial approximation is required to find the optimum point mathematically, since the maximum on a plane is at infinity unless the plane is perfectly horizontal. A horizontal plane probably would represent design points which happened to be on the same contour. This could be interpreted incorrectly as multiple optima inside the area of the contour, but should lead the analyst to further experimentation. Second-order approximations perform the experimentation in an organized manner.

RSM commonly uses factorial or fractional factorial designs for first-order approximations. To review, the suspected optimal point

becomes the center point of the factorial design. Then unit changes in each variable are defined on either side of their center point values. If unit changes are not appropriate for the actual analysis, then the changes can be coded. For example, suppose x_1 has a value of 20 at the center point. Values of 5 and 35 might be appropriate for experimenting in the area around the center point. First, note that the spacing needs to be the same on both sides (symmetrical). The values would be coded as -1, 0, and 1 for 5, 20, and 35 respectively. The coding assumes an appropriate scale for the variable, such as a ratio or interval scale. The variable is coded as follows:

$$x_1' = (x_1 - 20) / 15$$

The equation will be handy if the answer ends up being other than one of the coded values. This is entirely possible when finding the mathematical optimum of the second-order approximation.

The center point value should be checked before proceeding with the analysis. The center point is not part of the factorial design matrix for first-order analysis. The center point check may reveal that the surface is not represented properly by a plane in the region of experimentation. If the regression equation accurately predicts the center point response, increments up the steepest slope are taken until the errors between the predicted and experimental values of system response are too large. When this is the case, another first-order experiment is performed. If the slopes start to approach zero during this sequence, then the second-order model is required.

Second-Order Approximations

There are several considerations for second-order approximations that are not a problem with the first-order approximation. These considerations are the number of data points required and the orthogonality of the design.

The first-order approximations require only two levels for each factor; these are sufficient to define the required hyperplane in k-space. Factorial designs do not require excessive data points until considering 6 or 7 variables. Fractional factorial designs usually reduce the required number of data points with minor loss of information.

Second-order approximations require three levels of each variable to approximate the curved surface expected. With 4 factors, the full factorial design requires $3^4 = 81$ points to estimate 15 coefficients of the 4 main effects, 4 squared effects, 6 interaction terms, and the overall mean. Additional factors can help explain system responses, but geometrically increase the number of required replications. The analyst must consider the advantages and disadvantages of added factors. Fractional designs reduce the number of required replications, but not enough to offset the increase required by added factors.

Another consideration is the design of the experiment. Factorial and fractional factorial designs are orthogonal: desirable features for rotating the coordinate system and for determining the increments and direction for the method of steepest ascent. Orthogonality and efficient experimentation can be accomplished by central composite orthogonal designs and are recommended for analysis by numerous experts (8:343-348, 11:534,23:281,40:80,40:84,49:175).

Table 16 contains an example of a central composite orthogonal design matrix for 3 main effects, squared terms, and interaction terms.

x_1	x_2	x_3	x_1^2	x_2^2	x_3^2	x_1x_2	x_1x_3	x_2x_3
-1	-1	-1	1	1	1	1	1	1
1	-1	-1	1	1	1	-1	-1	1
-1	1	-1	1	1	1	-1	1	-1
1	1	-1	1	1	1	1	-1	-1
-1	-1	1	1	1	1	1	-1	-1
1	-1	1	1	1	1	-1	1	-1
-1	1	1	1	1	1	-1	-1	1
1	1	1	1	1	1	1	1	1
-1.682	0	0	2.828	0	0	0	0	0
1.682	0	0	2.828	0	0	0	0	0
0	-1.682	0	0	2.828	0	0	0	0
0	1.682	0	0	2.828	0	0	0	0
0	0	-1.682	0	0	2.828	0	0	0
0	0	1.682	0	0	2.828	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Table 16. 3^3 Central Composite Design (40:80)

The first three columns in this matrix correspond to three main effects, the next three represent squared terms, and the last three are the interaction terms. The values in the design matrix are coded as in the previous section. Great care is taken so the coded values are ±1 where possible to simplify the mathematics of matrix operations.

Normally, a column of 1's would precede this matrix, representing the mean. This is supplied automatically by the computer code provided at Appendix G. The first eight rows are the same as a 2^3 factorial design. The last six rows are center points.

Only one center point is required in a central composite design, but added points provide degrees of freedom for estimating error. Degrees of freedom also could be generated by replicating the entire design several times, but less efficiently. While most of the design points can be simulated using the same random number stream, this is not advisable for multiple center points. If the same random number stream is used for the different replications of the center point, identical answers will result, yielding a zero variance for the analysis. A zero variance might be misleading for stochastic processes.

The middle rows are axial points which are sometimes referred to as the "star." These points lie outside the regular polygon (for designs with more than 2 main effects) formed by the factorial design as indicated in Figure 4.

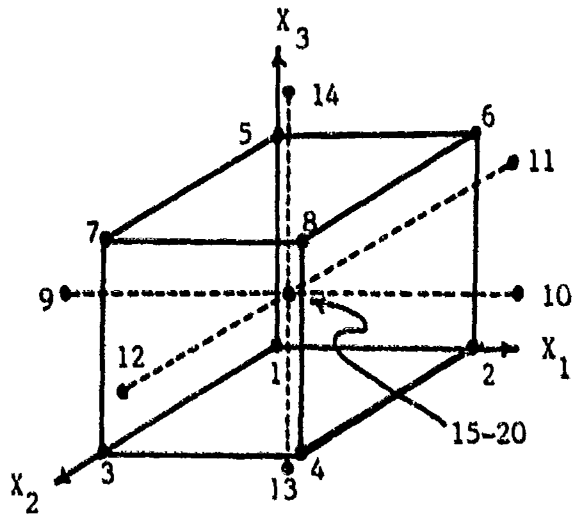

Figure 4. Star Plus Center Points

A particular number which preserves the orthogonality of the design is 1.682, calculated as follows (8:347):

$$2^{(k/4)} \quad (= 1.682 \text{ for } k = 3)$$

where k is the number of main effects in the central composite design.

For a half-fraction central composite design, replace the "k" with "k-1."

As can be seen from Figure 4, the axial points expand the area of experimentation to provide greater accuracy for the second-order approximation. This is accomplished along the primary axes only, instead of all combinations as in the factorial design. This reduces the required number of points from 3^k to $2^k + 2k$ + center points. This can be a very substantial savings. For the example above, this reduces the required runs from 81 to 20, with the added benefit of estimation of experimental error.

In Table 16, 2.828 is calculated by squaring 1.682.

The last point to discuss is how coded variables may be decoded. The following equation shows how 1.682 is decoded to a real-life value. All of the entries in a column are decoded with the same equation. For the previous example, the real-life values of the experiment are:

$$\begin{array}{ll} x_1 = 20 + 15x_1' & (= 45.23, \ -5.23 \text{ for } x_1' = \pm 1.682) \\ & (= 35.00, \ \ 5.00 \text{ for } x_1' = \pm 1) \end{array}$$

With this design, the analyst performs experimentation or simulations to gather the y-values (dependent variables). This is done by decoding the x-values and performing the experiments, one for each row of the design matrix. For the design presented in Table 16,

$$x_1 = 20 + 15x_1'$$

$$x_2 = 5 + 2.5x_2'$$

$$x_3 = 175 + 125x_3'$$

Experiment Number	Coded Values x_1'	x_2'	x_3'	Actual Values for Experiment x_1	x_2	x_3
1	-1	-1	-1	5	2.5	50
2	1	-1	-1	35	2.5	50
3	-1	1	-1	5	7.5	50
4	1	1	-1	35	7.5	50
5	-1	-1	1	5	2.5	300
6	1	-1	1	35	2.5	300
7	-1	1	1	5	7.5	300
8	1	1	1	35	7.5	300
9	-1.682	0	0	-5.23*	5	175
10	1.682	0	0	45.23	5	175
11	0	-1.682	0	20	0.795	175
12	0	1.682	0	20	9.205	175
13	0	0	-1.682	20	5	-35.25*
14	0	0	1.682	20	5	385.25
15	0	0	0	20	5	175
16	0	0	0	20	5	175
17	0	0	0	20	5	175
18	0	0	0	20	5	175
19	0	0	0	20	5	175
20	0	0	0	20	5	175

* assumes that negative values have physical meaning -- pick a smaller value for increment of x_1 and x_3 if this is not the case.

Table 17. Coded and Actual Independent Variable Values

20 experiments would be conducted as summarized in Table 17. Possible coding formulas for x_2 and x_3 are shown at the top of Table 17. Note that the experiment points are identified by number on Figure 4.

With the y-values, a computer program determines the regression coefficients. The analysis in the next chapter demonstrates the use of such a program.

The next step is to solve the second-order polynomial for an optimum system response. The optimum response is represented as a set of x-coordinates. The y-value is calculated based on the x-coordinates. The origin of the coordinate system is translated to this optimum set of x-coordinates and rotated to align with the contours of the response surface. Similar to the method of steepest ascent, further experimentation should be performed in the immediate vicinity of the stationary point to validate the surface fit.

Higher-Order Approximations

Current literature suggests that second-order approximations suffice for virtually all realistic problems. However, in rare instances, the need might arise for third-order approximation techniques. The concepts are exactly the same as for second order. Many more observations will be required with many more regression coefficients. After completing the numerous observations required for third- and higher-order curve fits, the resulting regression equations are difficult to relate to the real-world problem. For example, the third-order polynomial with three independent variables and full interactions is as follows:

$$
\begin{aligned}
y = b_0 &+ b_1x_1 + b_2x_2 + b_3x_3 \\
&+ b_{11}x_1^2 + b_{22}x_2^2 + b_{33}x_3^2 + b_{12}x_1x_2 + b_{13}x_1x_3 + b_{23}x_2x_3 \\
&+ b_{111}x_1^3 + b_{222}x_2^3 + b_{333}x_3^3 + b_{112}x_1^2x_2 \\
&+ b_{122}x_1x_2^2 + b_{113}x_1^2x_3 + b_{133}x_1x_3^2 + b_{223}x_2^2x_3 \\
&+ b_{233}x_2x_3^2 + b_{123}x_1x_2x_3
\end{aligned}
$$

Search for Optimal Point

The search for the optimum of a response function can have several results. Figure 5 summarizes the possibilities for two independent variables. The easiest case to interpret is the hill (or depression for minimization problems). However, a saddle point is possible. This implies two areas of optimal conditions. An example might be runway attacks where reciprocal attack headings could be expected to yield alternate optima. The saddle can be detected in the following manner. After rotation of the coordinate system, some of the slopes will be negative and some positive.

Other possibilities include ridges and slowly rising ridges. These provide the reason for preferring the method of steepest ascent for first-order analysis. Steepest ascent will climb directly up the ridge while the alternate technique — sectioning — deals with only one variable at a time. Sectioning will find the ridge, but may cross it at an angle. The "optimum" point thus identified may not be the true optimum, which would be found higher along the ridge line. In the case of a non-rising ridge, there are many optima, but sectioning only finds one of the multi-optimal solutions.

RSM Literature

This chapter is not intended to offer rigorous explanations of response surface methodology. Existing literature covers the field of RSM quite well. For quick overviews of response surface methodology, the interested reader should check Montgomery (38), Shannon (49), and Hicks (23). The innovator in the field of RSM was Box (5); his discussions are mathematically rigorous. For an excellent introduction

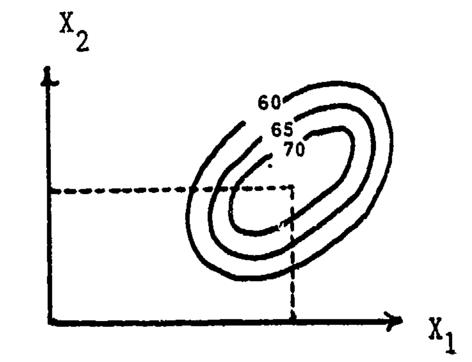

Fig. 5a. Maximum

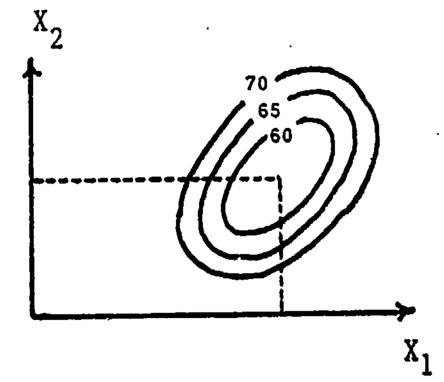

Fig. 5b. Minimum

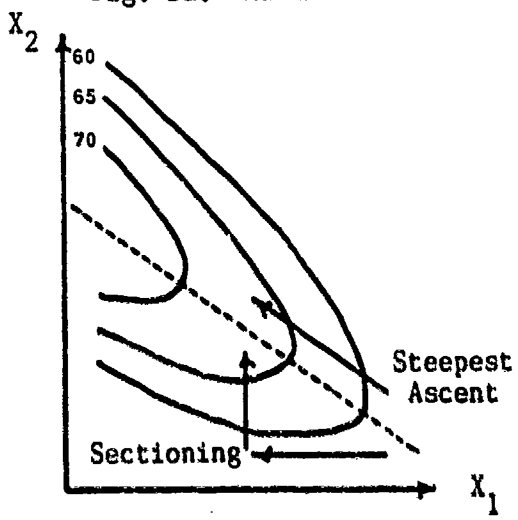

Fig. 5c. Rising Ridge

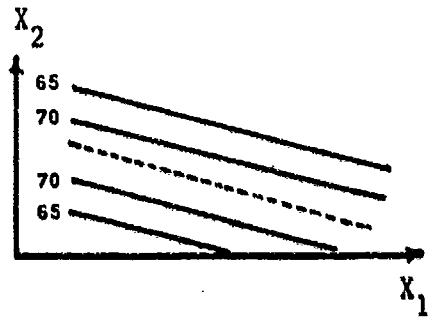

Fig. 5d. Stationary Ridge

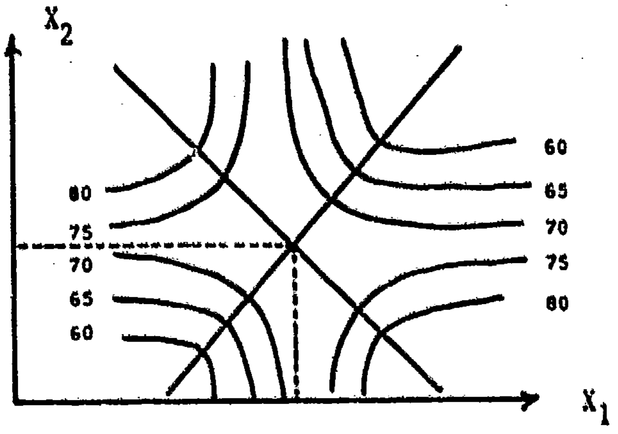

Fig. 5e. Saddle Point

Figure 5. Typical Response Contours

to RSM, Davies (11) offers a comprehensive treatment, although the British nomenclature may be hard to follow. The most sophisticated work is by Cochran and Cox (8). Other references cover the same material in more readable format. Myers' book — the only one specifically devoted to RSM — does a superior job of discussing the subject (40).

Shannon recommends Schmidt and Taylor (48) for a discussion of sectioning, though steepest ascent is preferred by most authors. Shannon also recommends Mirham (37) for application of RSM in simulation. Finally, Shannon cited Hill and Hunter's literature review, published in 1966 (24).

Application of RSM to Airfield Attack

Airfield attack analysis may be performed with the tools of response surface methodology. In order to use RSM, quantifiable and continuous variables are required. Many airfield attack parameters meet these criteria. RSM can transform a very complex, stochastic problem to a deterministic regression equation: a primary objective for this analysis.

The background material presented in this chapter outlines the basic framework for the latter portions of the airfield attack analysis. The next chapter documents the RSM applied to this analysis.

VII. Response Surface Experimentation

Variables for Analysis

Central composite designs suitable for response surface methodology require increasing numbers of design points as factors are added. Cochran and Cox (8) recommend full designs with 20 design points for 3 variables and 31 points for 4 variables. For more factors, half-fraction central composite designs are recommended. For 5 variables, 32 design points are required, and for 6 variables, 53 design points are needed. Each design point requires at least one run of 200 iterations during each analysis. Therefore, the number of input variables considered should be limited to the smallest practical value.

As indicated in Chapter V, seven potential airfield attack input parameters proved to be significant in affecting system responses. The responses considered were probability of runway cut and expected number of craters to fill to open a minimum clear TOL strip. Cochran and Cox (8) do not present a design for seven variables due to the excessive number of runs required by such a design. Due to limited computer resources and the envisioned use of AAPMOD on microcomputers, a smaller design was appropriate. The 4-variable full central composite design recommended by Cochran and Cox was adopted.

In order to fully explore system responses of the airfield attack scenario outlined in Chapter IV, groups of four variables were chosen. The initial analysis considers the most significant variables as indicated by the MANOVA conducted in Chapter V. This is followed by a parallel analysis which considers variables which aircrews can control.

This parallel analysis permits comparison of results between highly significant variables and controlled variables.

The most significant variables were: delivery error, weapon type, number of weapons, and weapon reliability. None of these are directly controlled by the aircrew. These variables were analyzed in one of the two 4-factor designs.

Weapon spacing and attack heading are among the least significant of the main effects chosen for the screening design. They are also the two which the aircrew can completely control. Weapon spacing and attack heading are significant to a high statistical level, but less significant than the variables highlighted in the preceding paragraph. Throughout the screening experiment, probability of arrival was the least significant of the input parameters. Consequently, the other 4-factor design used the following variables: delivery error, weapon reliability, attack heading, and weapon spacing.

In both designs, variables not considered were fixed at nominal levels. For a particular analysis, summary tables show the values of fixed variables. In this case, the high and low values are the same.

First-Order Approximation

As indicated in Chapter VI, initial application of response surface methodology assumes linear conditions, implicitly assuming that the center point of the analysis is somewhat removed from the optimum and the hyperplane describing the response surface is flat.

A fractional factorial design was run consistent with Chapter VI of this report. Tables 18, 19 and 20 summarize the values of the input variables, the design matrix, and the results. The design matrix in

Factor	Code Letter	High Value (+)	Low Value (-)
Pr(arrival)	A	0.955	0.955
Weapon type	B	Mk-84	Mk-82
Delivery error std dev	C	300/150	100/50
Number of weapons	D	12	6
Weapon reliability	E	0.99	0.59
Attack heading	F	155°	155°
Weapon spacing	G	75 ft	75 ft

Table 18. First-Order RSM Variable Values

Design Point	B	C	D	E
1	-	-	-	-
2	+	+	-	-
3	-	-	+	+
4	+	-	+	-
5	+	-	-	+
6	-	+	+	-
7	-	+	-	+
8	+	+	+	+

Table 19. First-Order RSM Design Matrix

Table 19 is depicted with "+" indicating the high value of a particular variable; "-" denotes the low value. Probabilities of cut and expected number of craters to fill are summarized in Table 20 for individual runways and both runways combined. Regression coefficients for the different dependent variables were calculated with the computer program at Appendix G. The regression coefficients are shown in Table 21.

For the other portion of the first-order RSM analysis, weapon type and number were fixed. There are numerous possible combinations of type and number of weapons. For the analysis, 6 Mk-84's and 12 Mk-82's were

Design Point	Probability of Cut R/W 1	R/W 2	Combined	Expected Craters to Fill R/W 1	R/W 2	Combined
1	0.340	0.550	0.215	0.355	0.620	0.215
2	0.100	0.465	0.065	0.021	0.035	0.017
3	0.980	0.910	0.895	2.550	1.415	1.310
4	0.905	0.865	0.780	1.685	1.290	0.950
5	0.875	0.990	0.805	1.925	2.225	1.500
6	0.775	0.445	0.135	0.305	0.525	0.140
7	0.195	0.530	0.085	0.250	0.635	0.090
8	0.740	0.950	0.700	1.600	2.010	1.175

Table 20. First-Order RSM Dependent Variable Values

Factor	Probability of Cut R/W 1	R/W 2	Combined	Expected Craters to Fill R/W 1	R/W 2	Combined
Mean response	0.551	0.713	0.468	1.086	1.094	0.677
Weapon type	0.104	0.104	0.135	0.221	0.296	0.236
Delivery error	-0.224	-0.116	-0.221	-0.542	-0.293	-0.319
# weapons	0.174	0.079	0.160	0.449	0.216	0.219
Reliability	0.146	0.132	0.169	0.495	0.477	0.344

Table 21. First-Order RSM Regression Coefficients

chosen as combinations of interest, because these loads can be carried by selected tactical aircraft (See Chapter V, page 57). The goal was to contrast results for a lesser number of heavy bombs with a greater number of lighter bombs. Separate analyses were performed with weapon type and number fixed at 6 Mk-84's, and then at 12 Mk-82's. The analyses fit the following variables: delivery error, weapon reliability, weapon spacing, and attack heading. Table 22 summarizes high and low values for the last four variables. Table 23 depicts the design matrix with updated factor identifiers. Tables 24-27 summarize the results for

Factor	Code Letter	High Value (+)		Low Value (-)
Pr(arrival)	A	0.950		0.950
Weapon type	B	Mk-84	-- or ---	Mk-82
Delivery error std dev	C	300/150		100/50
Number of weapons	D	12	-- or ---	6
Weapon reliability	E	0.99		0.59
Attack heading	F	170°		140°
Weapon spacing	G	100 ft		50 ft

Table 22. First-Order RSM Variable Values

Design Point	C	E	F	G
1	-	-	-	-
2	+	+	-	-
3	-	-	+	+
4	+	-	+	-
5	+	-	-	+
6	-	+	+	-
7	-	+	-	+
8	+	+	+	+

Table 23. First-Order RSM Design Matrix

6 Mk-84's and 12 Mk-82's. Tables 24 and 25 summarize Mk-82 results. Tables 26 and 27 summarize Mk-84 results using the same table layout as Tables 24 and 25. Both sets of tables are analogous to Tables 20 and 21, discussed earlier in the chapter.

Close inspection of Tables 24 and 26 revealed that the probability and expected number of craters were greater for the Mk-82's. Intuitively, Mk-84's with weapon spacing set at the same level as Mk-82's should have produced the larger system responses. The most likely explanation is the interacting effect of delivery error and stick

Design Point	Probability of Cut R/W 1	R/W 2	Combined	Expected Craters to Fill R/W 1	R/W 2	Combined
1	0.520	0.615	0.335	0.585	0.720	0.340
2	0.360	0.665	0.240	0.445	0.950	0.260
3	0.710	0.270	0.210	1.145	0.275	0.210
4	0.075	0.450	0.035	0.080	0.530	0.035
5	0.045	0.160	0.005	0.045	0.160	0.005
6	0.640	0.940	0.605	1.375	1.805	0.895
7	0.520	0.630	0.330	0.520	0.665	0.330
8	0.335	0.565	0.190	0.590	0.615	0.205

Table 24. First-Order RSM Dependent Variable Values (Mk-82)

Factor	Probability of Cut R/W 1	R/W 2	Combined	Expected Craters to Fill R/W 1	R/W 2	Combined
Mean response	0.401	0.537	0.244	0.598	0.715	0.285
Delivery error	-0.197	-0.077	-0.126	-0.308	-0.151	-0.159
Reliability	0.063	0.163	0.098	0.134	0.294	0.138
Attack heading	0.039	0.019	0.016	0.199	0.091	0.051
Weapon Spacing	0.002	-0.131	-0.060	-0.023	-0.286	-0.098

Table 25. First-Order RSM Regression Coefficients (Mk-82)

Design Point	Probability of Cut R/W 1	R/W 2	Combined	Expected Craters to Fill R/W 1	R/W 2	Combined
1	0.360	0.645	0.220	0.380	0.885	0.225
2	0.180	0.455	0.075	0.205	0.660	0.075
3	0.385	0.380	0.150	0.445	0.400	0.150
4	0.040	0.265	0.015	0.040	0.320	0.015
5	0.050	0.225	0.020	0.055	0.225	0.020
6	0.555	0.905	0.500	0.905	1.820	0.695
7	0.795	0.705	0.545	0.995	0.835	0.565
8	0.195	0.545	0.090	0.235	0.645	0.095

Table 26. First-Order RSM Dependent Variable Values (Mk-84)

Factor	Probability of Cut R/W 1	R/W 2	Combined	Expected Craters to Fill R/W 1	R/W 2	Combined
Mean response	0.320	0.516	0.202	0.408	0.724	0.230
Delivery error	-0.204	-0.143	-0.152	-0.274	-0.261	-0.179
Reliability	0.111	0.137	0.101	0.178	0.266	0.128
Attack heading	-0.026	0.008	-0.013	-0.001	0.073	0.009
Weapon spacing	0.036	-0.052	-0.001	0.025	-0.198	-0.023

Table 27. First-Order RSM Regression Coefficients (Mk-84)

length. Mk-84 sticks miss enough of the runway enough of the time to yield lower overall system responses.

The method of steepest ascent was applied to the Mk-82 experiment. Table 28 shows the calculations required to determine the steepest ascent direction and the number of steps possible before reaching the bounds of the system. The limiting factor for both Mk-82's and Mk-84's was weapon reliability. Further steps would have exceeded a weapon reliability of 1.00 which is physically impossible.

Only Mk-82 results are depicted in Table 28. The first row contains the regression coefficients output from the computer program given at Appendix G. The second row shows the step size indicated in Table 22. The next row contains the product of the first two rows for each variable. In this way, proportionality is maintained among the variables. Next, the variables are scaled so that delivery error will have an incremental step size of 25 feet. This is done by dividing each element of row 3 by a value of 4.0. The next row contains the original values of the uncoded variables. The increment rows are calculated by adding row 4 to the initial row successive times for each increment.

	C	E	F	G	$\hat{y}$	y_{actual}
Regression coefficient	-0.126	0.098	0.016	-0.060		
Step size	100 ft	0.200	15°	25 ft		
Proportional step	-12.63	0.020	0.245	-1.5		
Change for ΔC = 25	-25	0.039	0.484	-2.97		
In coded units	-0.125	0.193	0.032	-0.119		
Initial value	200/100	0.79	155°	75 ft	0.244	0.360
Increment 1	175/87.5	0.83	155.5	72.03	0.286	
2	150/75	0.87	156.0	69.06	0.328	
3	125/62.5	0.91	156.5	66.09	0.370	
4	100/50	0.94	156.9	63.12	0.412	
5	75/73.5	0.98	157.4	60.16	0.454	

Table 28. Steepest Ascent Increments

The increase in the response for each increment is found by substituting the coded increments from row 5 into the regression equation while ignoring the mean.

The method of steepest ascent proceeds with the assumption of linearity until the difference between the predicted y-value and the experimental value is greater than the error tolerance chosen by the analyst. When the center point was checked, there was a 48 percent error between the predicted value of 0.244 and the actual value of 0.36. Therefore, the first order design did not represent the system responses within the area of experimentation.

There was a choice between limiting the scope of a first-order search or proceeding directly to the second-order approximation. The scope of the independent variables could have been reduced drastically. If a small enough area is considered, the linear assumption becomes more accurate. This was undesirable in this analysis: the original ranges

for the input variables were picked because they were of interest. Narrowing the scope of these variables would have been counter to the purpose of the analysis.

Second-Order Approximation

Since the first-order approximation inadequately fit the system responses, the second-order approximation was adopted. Table 29 summarizes the independent variables for the second-order analysis. Separate analyses were run for 6 Mk-84's and 12 Mk-82's. Table 30 shows the design matrix for the second-order analysis with four independent variables. Table 31 summarizes the results of the runs indicated by the design matrix in Table 30.

Both regression and lack of fit were significant at any confidence level chosen. The analysis of variance is shown in Table 32. The statistical significance of lack of fit indicates that the range of input parameters had to be reduced. This was accomplished in the follow-on second-order RSM experiment. The stationary point appears to be a local minimum. The smaller area of consideration in the follow-on experiment led to identifying a possible local maximum.

The stationary point is indicated in Table 33. Note that the independent variable values are within the ranges used in the experiment, but that a local minimum may have been found. The status of the indicated stationary point is unknown pending further testing. Since surrounding design points resulted in larger system responses, a local minimum seems to be indicated.

Effect	-2	-1	0	+1	+2
Delivery error std dev	0/0	150/75	300/150	450/225	600/300
Weapon reliability	0.59	0.69	0.79	0.89	0.99
Attack heading	90.0	112.5	135.0	157.5	180.0
Weapon spacing	0	40	60	80	100

Table 29. Initial Second-Order RSM Variable Coding

Design Point	C	E	F	G	C^2	E^2	F^2	G^2	CE	CF	CG	EF	EG	FG
1	-1	-1	-1	-1	1	1	1	1	1	1	1	1	1	1
2	1	-1	-1	-1	1	1	1	1	-1	-1	-1	1	1	1
3	-1	1	-1	-1	1	1	1	1	-1	1	1	-1	-1	1
4	1	1	-1	-1	1	1	1	1	1	-1	-1	-1	-1	1
5	-1	-1	1	-1	1	1	1	1	1	-1	1	-1	1	-1
6	1	-1	1	-1	1	1	1	1	-1	1	-1	-1	1	-1
7	-1	1	1	-1	1	1	1	1	-1	-1	1	1	-1	-1
8	1	1	1	-1	1	1	1	1	1	1	-1	1	-1	-1
9	-1	-1	-1	1	1	1	1	1	1	1	-1	1	-1	-1
10	1	-1	-1	1	1	1	1	1	-1	-1	1	1	-1	-1
11	-1	1	-1	1	1	1	1	1	-1	1	-1	-1	1	-1
12	1	1	-1	1	1	1	1	1	1	-1	1	-1	1	-1
13	-1	-1	1	1	1	1	1	1	1	-1	-1	-1	-1	1
14	1	-1	1	1	1	1	1	1	-1	1	1	-1	-1	1
15	-1	1	1	1	1	1	1	1	-1	-1	-1	1	1	1
16	1	1	1	1	1	1	1	1	1	1	1	1	1	1
17	-2	0	0	0	4	0	0	0	0	0	0	0	0	0
18	2	0	0	0	4	0	0	0	0	0	0	0	0	0
19	0	-2	0	0	0	4	0	0	0	0	0	0	0	0
20	0	2	0	0	0	4	0	0	0	0	0	0	0	0
21	0	0	-2	0	0	0	4	0	0	0	0	0	0	0
22	0	0	2	0	0	0	4	0	0	0	0	0	0	0
23	0	0	0	-2	0	0	0	4	0	0	0	0	0	0
24	0	0	0	2	0	0	0	4	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 30. Second-Order Four-Factor RSM Design Matrix

Design Point	P_{cut}	Design Point	P_{cut}
1	0.345	16	0.095
2	0.020	17	0.185
3	0.535	18	0.010
4	0.065	19	0.020
5	0.405	20	0.275
6	0.025	21	0.080
7	0.520	22	0.025
8	0.040	23	0.040
9	0.040	24	0.075
10	0.010	25	0.110
11	0.225	26	0.110
12	0.025	27	0.125
13	0.265	28	0.128
14	0.015	29	0.127
15	0.580	30	0.123
		31	0.126

Table 31. Combined Probability of Runway Cut for Initial Second-Order RSM Design

A N A L Y S I S O F V A R I A N C E

Source	SS	d.f.	MS	F	p
Regression	0.6107	14	0.0436	3.54	0.0088
Lack of fit	0.1953	10	0.0195	64.34	0.0
Error	0.0018	6	0.0003		
Total	0.8078	30			

Table 32. ANOVA for Goodness of Fit for Initial Second-Order RSM

Effect	Coded Value	Actual Value
Delivery error std dev	1.7880	568.2/284.1
Weapon reliability	0.1137	0.80
Attack heading	-1.3672	104.20
Weapon spacing	0.9600	79.20
Probability of cut	-0.0067	-0.0067

Table 33. Stationary Point Analysis for Initial Second-Order RSM

The Mk-82 regression equation developed in the initial second-order analysis for the probability of cut for the combination of runways was:

$$
\begin{aligned}
y = 0.126 &- 0.124\,C + 0.061\,E + 0.024\,F - 0.026\,G \\
&+ 0.012\,C^2 + 0.024\,E^2 + 0.0003\,F^2 + 0.002\,G^2 \\
&- 0.041\,CE - 0.036\,CF + 0.043\,CG + 0.006\,EF \\
&+ 0.014\,EG + 0.039\,FG
\end{aligned}
$$

where C, E, F, and G are coded values for the main effects, and

y = predicted probability of cut
C = delivery error standard deviation
E = weapon reliability
F = attack heading
G = weapon spacing

The response surface represented by the equation has a stationary point as identified in Table 33. Since this stationary point appeared to be a local minimum, further experimentation was required. The next section describes the follow-on second-order experiment.

cond-Order Approximation of Reduced Surface

A visual inspection of system responses was the most direct method of narrowing the area for consideration. The design points were

inspected to see where relative maxima occurred. This inspection was the basis for the updated ranges of variables found in Table 34.

The smaller scope of the independent variables increased the likelihood of a good second-order surface fit in the region of interest. The further experimentation proved more successful. The stationary point for Mk-82's appeared to be a local maximum, but the stationary point for Mk-84's appeared to be a local minimum. Both optima were infeasible, with their respective response surfaces extending beyond the feasible region of the airfield attack problem. The surface fit was statistically significant and valid. The analyst must supply the boundary conditions which limit the response surface to a feasible region. Restriction of the response surface to a feasible region will be addressed in Chapter VIII.

The experiment design matrix remains exactly the same as before. The probabilities of cut for combined runways are the results of the experimental runs shown in Table 35.

The system responses from Table 35 indicated the center point was closer to the optimal point than the initial second-order estimate. The center point replications appeared to have a larger variance than before, which drove the p-values for lack of fit to larger values. Regression results for data in Table 35 are summarized in Table 36.

The p-value of 0.0265 indicates that lack of fit is still significant at a level of 0.05. Some variations in system response are not explained by the regression equation implied by Table 36. The conclusions drawn from the analysis must be verified by experimentation. The stationary point is indicated in Table 37.

Effect	-2	-1	0	+1	+2
Delivery error std dev	0/0	75/37.5	150/75	225/112.5	300/150
Weapon reliability	0.79	0.84	0.89	0.94	0.99
Attack heading	135.0	146.3	157.5	168.7	180.0
Weapon spacing	60	70	80	90	100

Table 34. Follow-on Second-Order RSM Variable Coding

Design Point	P_{cut}	Design Point	P_{cut}	Design Point	P_{cut}
1	0.430	11	0.355	21	0.330
2	0.335	12	0.320	22	0.165
3	0.600	13	0.510	23	0.645
4	0.375	14	0.210	24	0.405
5	0.630	15	0.590	25	0.580
6	0.270	16	0.340	26	0.500
7	0.700	17	0.435	27	0.550
8	0.270	18	0.290	28	0.495
9	0.255	19	0.445	29	0.490
10	0.190	20	0.630	30	0.570
				31	0.520

Table 35. Combined Probability of Runway Cut for Follow-on Second-Order RSM Design

A N A L Y S I S O F V A R I A N C E

Source	SS	d.f.	MS	F	p
Regression	0.5812	14	0.0415	8.08	0.00008
Lack of fit	0.0739	10	0.0074	5.33	0.0265
Error	0.0083	6	0.0014		
Total	0.6634	30			

Table 36. ANOVA for Goodness of Fit for Follow-on Second-Order RSM (Mk-82)

The Mk-82 regression equation in the follow-on second-order analysis for the probability of cut for the combination of runways was:

$$\begin{aligned} y = 0.529 &- 0.085\,C + 0.045\,E + 0.014\,F - 0.055\,G \\ &- 0.045\,C^2 - 0.001\,E^2 - 0.074\,F^2 - 0.004\,G^2 \\ &- 0.008\,CE - 0.058\,CF + 0.029\,CG - 0.010\,EF \\ &+ 0.010\,EG + 0.025\,FG \end{aligned}$$

where C, E, F, and G are coded values for the main effects, and

y = predicted probability of cut
C = delivery error standard deviation
E = weapon reliability
F = attack heading
G = weapon spacing

Effect	Coded Value	Actual Value
Delivery error std dev	-3.2096	-90.72/-45.36
Weapon reliability	11.3414	1.46
Attack heading	-0.1812	155.46
Weapon spacing	-4.4635	35.34
Probability of cut	1.0454	1.0454

Table 37. Stationary Point Analysis for Follow-on Second-Order RSM (Mk-82)

Once again the stationary point lies in an infeasible region, since negative delivery errors and probabilities in excess of 1.0 are infeasible. The analyst must apply the boundary conditions which will keep the experiment in a feasible region of the system responses. (See Chapter VIII.)

The analysis of variance and stationary point results for Mk-84's are shown in Tables 38 and 39. For the first time, lack of fit was

ANALYSIS OF VARIANCE

Source	SS	d.f.	MS	F	p
Regression	1.1096	14	0.0793	21.99	0.0000
Lack of fit	0.0499	10	0.0050	3.86	0.0560
Error	0.0078	6	0.0013		
Total	1.1673	30			

Table 38. ANOVA for Goodness of Fit for Follow-on Second-Order RSM (Mk-84)

Effect	Coded Value	Actual Value
Delivery error std dev	25.2622	2045/1022
Weapon reliability	2.4511	1.01
Attack heading	1.2563	171.63
Weapon spacing	11.8698	198.70
Probability of cut	-1.9638	-1.9638

Table 39. Stationary Point Analysis for Follow-on Second-Order RSM (Mk-84)

statistically insignificant as indicated by the p-value greater than 0.05 in Table 38. However, lack of fit would have been significant if 0.10 had been the desired level of significance. A p-value in the range 0.20 to 0.40 would have indicated strong likelihood of correctly assuming lack of fit to be insignificant. The chance for error is large if lack of fit is assumed insignificant based on the p-value of 0.0560 because it is so close to the chosen level of significance. Therefore,

the regression equation for Mk-84's can be used, but should be checked against further experimentation when stationary points are identified.

Once again, the calculated stationary point was in an infeasible region. Weapon reliability was larger than 1.0, and the predicted response was less than zero. The predicted probability of cut appeared to be a local minimum. The analyst must apply boundary conditions to constrain the analysis to a feasible region. Such constraints may generate a local maximum.

The Mk-84 regression equation in the follow-on second-order analysis for the probability of cut for the combination of runways was:

$$\begin{aligned} y = 0.394 &- 0.197\,C + 0.059\,E - 0.033\,F + 0.013\,G \\ &- 0.001\,C^2 + 0.009\,E^2 - 0.033\,F^2 - 0.027\,G^2 \\ &- 0.006\,CE + 0.000\,CF + 0.024\,CG - 0.011\,EF \\ &+ 0.005\,EG + 0.012\,FG \end{aligned}$$

where C, E, F, and G are coded values for the main effects, and

y = predicted probability of cut
C = delivery error standard deviation
E = weapon reliability
F = attack heading
G = weapon spacing

Note: the coefficient for the CF interaction term was zero to the 13^{th} decimal place.

Summary

This chapter has applied response surface methodology to the airfield attack problem. Two first-order approximations were performed. These explored highly significant AAPMOD input parameters in one experiment and parameters controllable by the aircrew in another.

A check of the design center points indicated that the linear assumption required for first-order approximations was invalid. Rather than severely restrict the region of experimentation, second-order analysis was adopted.

The second-order experimentation concentrated on those variables controllable by the aircrew. Two weapons loads of 6 Mk-84 bombs and 12 Mk-82 bombs were established for analysis. Second-order RSM analysis of delivery error, weapon reliability, attack heading, and weapon spacing was accomplished for both of these weapons loads. The initial second-order region of experimentation was too large. The regression equations did not explain enough of the data to be conclusive. The area of interest was narrowed with improved results. The final second-order experiment for Mk-84's yielded a marginally valid response surface fit and corresponding regression equation, although the stationary point appeared to be a local minimum. In all cases, the regression equations contained stationary points which were not feasible. In two out of three cases, at least one independent variable value at the supposed stationary point was infeasible. In those cases, the analyst would be required to apply boundary conditions to maintain realism.

Several important conclusions resulted from this analysis. The scope of airfield attack parameters was too large to permit effective use of response surface methodology. The ranges of parameters could have been reduced, and optimal operating conditions could have been found. However, the search for optima requires very narrow ranges of input parameters to produce reasonable response surface fits. The limited scope of a particular response surface fit offers little flexibility in attack planning. Aircrews must choose attack headings

to avoid hostile fire. Additionally, weapon spacing is restricted by equipment capabilities and other factors. Therefore the calculated optimum conditions may not be attainable. For the methodology to be of use, the response surface must represent a broad range of input parameters.

In this analysis, the response appeared to undulate to such an extent that a quadratic surface fit over broad ranges of the input variables was not possible. This was checked with several test cases: simulation runs at the fitted design points predicted system response accurately within a 10 percent error tolerance. Simulation runs at points outside the region of experimentation produced prediction errors of up to 50 percent.

Suboptimization of input parameters directly controlled by the aircrew may be required. In the one experiment that appeared to yield a local maximum, attack heading and weapon spacing were both feasible, while delivery error and weapon reliability had driven to infeasible values. Smaller delivery errors and higher weapon reliability should produce higher probabilities of cut. Though infeasible in this case, the variables moved in the expected direction. Since delivery error and weapon reliability are largely uncontrolled, these should be set at reasonable values, permitting weapon spacing and attack heading to be suboptimized. This is a primary topic in the next chapter.

VIII. Applications

Two general approaches to optimizing airfield attack parameters are on-the-spot computer simulation of airfield attack plans and mathematical optimization schemes.

Computer Simulation

Computerized attack assessment offers the advantages of flexibility and responsiveness, but is hampered by relatively long execution times and cost. Consequently, some care must be taken when organizing attack assessment in this manner. The Attack Assessment Program--MODIFIED (AAPMOD) has been designed to fulfill the requirement of validated attack assessment for field use.

The most efficient use of AAPMOD would be to have available preprogrammed input databases containing target, weapon, and attack pattern information. With these databases, the user could formulate attacks quickly and analyze them. A single run could then be formulated and run in less than one hour in many cases. Wing mission-planning officers could run such programs upon receipt of a fragmentary order and prior to aircrew briefings. Aircrews could use the programs to improve tactics by analyzing proposed plans during peacetime training to determine maximum effectiveness.

The User Manual at Appendix A is a comprehensive, step-by-step guide for use of the AAPMOD series of programs. The database loader programs have been designed with the end user in mind; hence plenty of on-screen help is available and extensive error-checking of inputs is accomplished.

Use of AAPMOD for current analyses permits maximum use of the considerable flexibility of this program. Airfield attack parameters can interact in countless combinations. The fragmentary order for mission tasking fixes many of the parameters at specific levels, leaving fewer parameters subject to aircrew judgement.

A possible method of employing on-the-spot computer simulation would be to generate several potential attack plans for a given target complex. Attack plans would correspond to the design points in the RSM designs. Each plan could be programmed in approximately 30 minutes and run on microcomputers using the instructions provided at Appendix A. The plan providing the best relative system response would provide maximum effectiveness for a given level of hostile fire. The number of data points would be very small compared to RSM analysis. These points hardly could be expected to yield the global optimum, but near-optimal results are likely. On a daily basis, aircrews apply heuristics in planning airfield attacks. Presumably, these heuristics would be applied to airfield attack plans submitted for analysis. The process would be one of choosing the best near-optimal plan.

AAPMOD computer simulation could be used in peacetime training in a similar manner. Teams could compete to determine the best attack plan. In the competitive environment of a fighter squadron, it would not take long for the aircrews to develop effective planning techniques and learn from each other's mistakes. This, in turn, would be extended to wartime operations, provided that the aircrews gained confidence in AAPMOD results.

Mathematical Optimization

An alternative approach to computer simulation is to conduct extensive analysis of enemy airdromes prior to the outbreak of hostilities. A response surface can be fitted to the particular airdromes of interest. Within the limits of feasibility, optima can be determined from the regression equation which characterizes the response surface.

Aircrews can control attack heading and weapon spacing. The other airfield attack parameters can be fixed at nominal levels based on historical data, gunnery records, and standard weapon loads. In this way AAPMOD can be used to sub-optimize attack heading and weapon spacing.

The coding equations for the airfield attack parameters are:

$$C = (C' - 150) / 75 \qquad E = (E' - 0.89) / 0.05$$

$$F = (F' - 157.5) / 11.25 \qquad G = (G' - 80) / 10$$

where the primed values are real-world values, the unprimed values are coded values, C is the delivery error standard deviation, E is the weapon reliability, F is the attack heading, and G is the weapon spacing.

The regression equation for Mk-82 attack of the airfield depicted in Figure 1 is repeated here:

$$\begin{aligned} y = 0.5293 &- 0.0854\,C + 0.0454\,E + 0.0138\,F - 0.0550\,G \\ &- 0.0449\,C^2 - 0.0012\,E^2 - 0.0737\,F^2 - 0.0043\,G^2 \\ &- 0.0075\,CE - 0.0575\,CF + 0.0288\,CG - 0.0100\,EF \\ &+ 0.0100\,EG + 0.0250\,FG \end{aligned}$$

where y is the predicted probability of cut; and C, E, F, and G are as described above.

For this regression equation, probability of arrival was fixed at 0.95, type of weapon was a Mk-82, number of weapons was 12. Uncontrolled variables in the equation are delivery error and weapon reliability. Both of these could be estimated by the squadron weapons officer. The choice for these parameters necessarily would be in the feasible region. This would avoid some of the problems noted in Chapter VII when calculating stationary points. As an example, delivery accuracy and weapons reliability might be determined to be 200/100 feet and 0.85 for a particular airfield attack with designated crews. These values can be substituted into the regression equation. This leaves one equation with two unknown variables. The optimal values of weapon spacing and attack heading with all other variables fixed are found by solving the two simultaneous equations that result when the simplified regression equation is differentiated with respect to weapon spacing and attack heading. The two partial derivatives would be as follows:

$$\frac{\partial y}{\partial F} = -0.0165 - 0.1474\,F + 0.0250\,G = 0$$

$$\frac{\partial y}{\partial G} = -0.0438 + 0.0250\,F - 0.0086\,G = 0$$

at stationary point

These equations are solved giving G = -10.688 and F = -1.9427. These are the coded values. The actual values for weapon spacing and attack heading would be -26.89 feet and 135.6°, respectively. A negative weapon spacing is infeasible. The analyst must bound the system in some manner. There are sophisticated mathematical tools for searching for optima which also meet constraints. These are beyond the scope of the typical aircrew in a flying squadron and were discounted.

A more simple means of bounding the problem is possible. The negative value for weapons spacing implies that small values increase the probability of cut. If this is the case, then the weapons spacing can be set at a minimum value in a manner similar to delivery error and weapon reliability. Suppose that 30 feet is the minimum possible weapons spacing. Substituting into the equation for $\partial y/\partial F$ results in a coded attack heading of 1.754. This is decoded as 177.2° -- almost directly along the primary runway axis. The results easily can be checked by using the original regression equation to find a predicted value of y and then running AAPMOD to find the experimental value. The predicted value of y is 0.074 and the experimental value is 0.120.

Several observations are appropriate. First, the indicated attack heading is not very appealing on an intuitive level. Second, the probability of cut is relatively poor. Finally, the center point of the underlying RSM analysis gave probabilities of cut averaging 0.530. The simplifying assumption of minimum weapon spacing was invalid.

The regression equation is valid in the region of experimentation, which should facilitate the search for optima. Different combinations of weapons spacing and attack heading can be checked with the regression equation for the best system response. This is an informal method of searching the response surface for optimal operating conditions. Table 40 summarizes the results of this search. The original center point was a reasonable estimate.

Another possible drawback of this technique is the difficulty of anticipating all the parameter levels which must be fixed. The scenario used to generate the previous example was somewhat restrictive. Six aircraft attacked a simplified airfield using the same weapons, attack

			ATTACK HEADING		
			140°	160°	180°
WEAPON	SPACING	40 ft	0.529	0.496	0.000
		60 ft	0.415	0.471	0.062
		80 ft	0.267	0.412	0.090
		100 ft	0.084	0.318	0.087

Table 40. Search for Constrained Optimum Operating Conditions

patterns, and attack heading. The combinations of these variables might require numerous such scenarios for mathematical optimization to be effective.

Mathematical optimization would be accomplished more easily with a further simplified scenario. This was the approach pursued by Peck (41). Figure 6 depicts a typical chart from Peck's analysis. This is based on a single 4000 foot long strip. The object of an attack is to produce one cut which will close the strip. From one to six passes are considered. Peck found that individual attacks could be treated as independent with minor loss of accuracy in limited cases.

Summary

Mathematical suboptimization has been stymied by inadequate response surface fit and complex mathematical methods beyond the average aircrew. Under certain circumstances, heuristics seem to produce results better than mathematically derived optima.

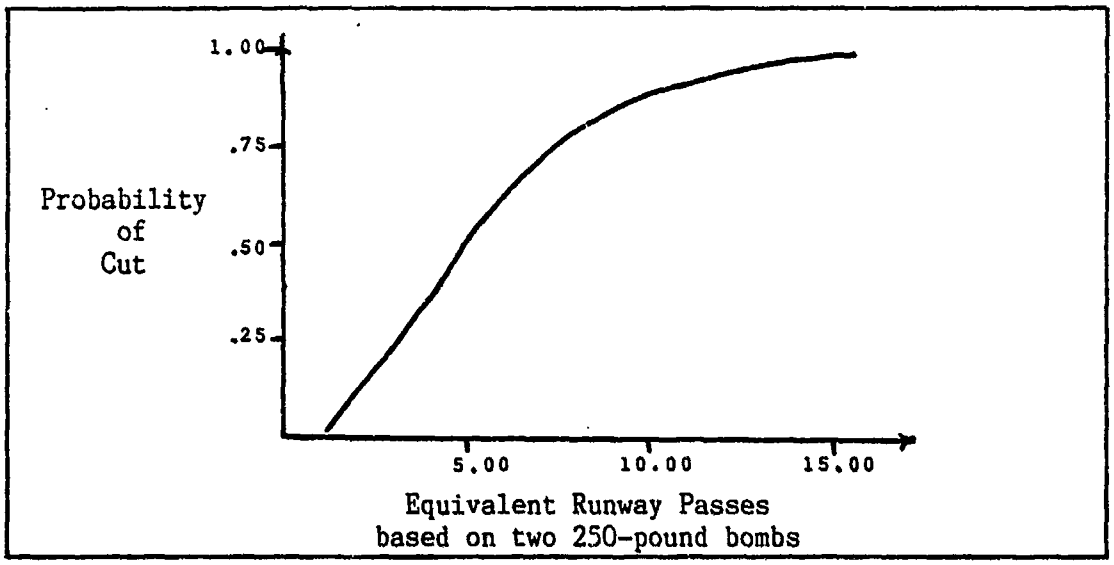

Figure 6. Airfield Attack Effectiveness

It is important to reiterate that response surface methodology is capable of identifying a local maximum response. The search for this optimum would cover successive sections of response surface small enough to produce an adequate fit. Once identified, the local maximum and the accompanying regression equation would be restricted to the small area. If that area were forbidden to the aircrew by hostile gun emplacements or weapon requirements, the prediction would be of little use. The regression equation would not be likely to make accurate predictions outside the restricted area of experimentation.

In addition, the response surface could be invalidated by any changes to the target airfield; numerous runs would have to be reaccomplished for each change.

For these reasons, mathematical optimization and response surface methodology are not recommended for this problem, except to identify areas of near-optimality useful in developing aircrew heuristics.

The recommended approach is use of on-the-spot computer simulation. The user manual for the implementation of microcomputer versions of AAPMOD is located at Appendix A. PASCAL computer code is located at Appendix B. Predefined databases can be maintained and updated easily for potential target airfields, weapons, and attack patterns. When time-critical mission tasking arrives at the squadron, several attack plans can be assembled and analyzed promptly.

IX. Sensitivity Analysis

Several areas of the analysis are sensitive to assumed distributions and parameters. Some sensitivity analysis has already been performed in the form of the regression coefficients for the various independent variables. The larger the regression coefficient, the more sensitive is the analysis to that parameter. Some judgement must be used in that the regression coefficient shows the effect of a unit change in the scaled variable. The scaling could be such that fairly major changes in the raw variable have relatively little effect on the system response.

There are other areas which might be sensitive to changes besides the independent variables in the problems. One area is the probability distribution chosen to model the aiming, delivery, and ballistic errors. The bivariate normal distribution is used to model ballistic and delivery errors. This is reasonable for large numbers of passes due to the Central Limit Theorem. The normal distribution is very widely used for this type of error throughout the analytical community. JMEMs make use of bivariate normal distributions although JMEMs were written assuming that range and deflection errors were independent. The normal distribution is so widely accepted for description of probable errors that it would be ill-advised to use any other distribution. The parameters of the delivery error distribution were varied during the analysis. The standard deviations for the ballistic error distributions were fixed throughout the analysis. These would have an effect similar to delivery error.

A triangular distribution is used to model the ability of aircrews to discern the desired aimpoint as designated by higher headquarters or by the aircrew during pre-mission planning. This needs to be considered because determining an aimpoint along a nondescript line feature is inherently difficult. Miglin's version of AAPMOD uses the mid-point as zero aiming error and the end points as ±1000 feet. This seems reasonable for line features which probably will be 8000 to 9000 feet in length. Still, some sensitivity analysis is warranted. The end point parameter in the triangular distribution was tested at 0, 500, 1000, 1500 and 2000 feet using one particular design point in which the 1000-foot end point had produced a combined probability of cut of 0.700. The design point chosen is already near optimum for all other variables which should make the response particularly sensitive to aimpoint determination errors. There was no change in combined probability of cut at 0 or 500 feet. The probability of cut dropped to 0.610 for 1500 feet and 0.510 for 2000 feet. It was concluded that AAPMOD was not particularly sensitive to the triangular distribution or the end point parameter. The 2000-foot figure implies that the aircrew cannot determine the midpoint of an 8000 foot long runway. This only would be the case in poor weather or against a well camouflaged runway.

AAPMOD generates normal random variates using the exact inverse method. This method can be time-consuming since sines, cosines, and natural logarithms must be calculated. An approximate method is discussed in Banks (3:318). This method was tested on the CDC Cyber for several representative runs. The average reduction in central processor time was 5.5 percent for the runs tested. The approximate method is not as accurate as the exact inverse method, suggesting that the exact

inverse generator be retained. The analyst should consider the accuracy lost for the relatively small gain in computer time.

Another reduction in required computer time was realized by not calculating expected area to fill to open a runway. Sensitivity of the analysis to this parameter was tested against 31 separate design points. In each case, the same relative ranking of design points was indicated by both the expected area to fill and the expected number of craters to fill. The latter figure is automatically supplied in the abbreviated output and requires approximately one-fifteenth the processor time. Calculation of expected area to fill is not recommended.

Release mode was not a variable considered for the main analysis. System sensitivity to paired releases will be discussed next. Paired releases resulted in more favorable system responses when low delivery errors and attack headings near-perpendicular to the runway were assumed. Paired releases were less favorable than single releases for attacks along the runway centerline. If delivery error were assumed to be large, the single releases dominated. Consequently, paired releases are indicated for accurate delivery systems (such as the F-16), but not for inaccurate systems.

The last area for sensitivity analysis is the number of aircraft participating in the attack. A simple test case was run in which from 1 to 12 aircraft attacked the airfield depicted in Figure 1. The system response decreases when fewer attacking aircraft participate in the attack. Table 41 summarizes the number of aircraft attacking a particular aimpoint as well as the system response. The expected number of craters to fill increased monotonically with increasing numbers of aircraft, but the probability of cut did not increase monotonically.

# Acft	Aimpoints (3000,0)	(6000,0)	(1500,2598)	P_{cut}	# Craters
1	1	0	0	0.000	0
2	1	0	1	0.000	0
3	1	1	1	0.160	0.160
4	2	1	1	0.225	0.225
5	2	2	1	0.340	0.340
6	2	2	2	0.700	0.815
7	3	2	2	0.775	0.985
8	3	3	2	0.770	0.995
9	3	3	3	0.920	1.640
10	4	3	3	0.910	1.690
11	4	4	3	0.940	1.795
12	4	4	4	0.980	2.495

Table 41. Sensitivity of Number of Attacking Aircraft

The probability of cut dropped when adding an eighth aircraft and when adding a tenth aircraft. This anomaly was attributed to random number streams at first. However, switching random number seeds generated similar results.

X. Concluding Remarks

Recommendations for Further Analysis

This analysis is only one step in the process of analyzing airfield attacks. There are many possibilities for further analysis.

1. Douglas (15) concluded that single aircraft always should attack near-perpendicular to the runway centerline for best probability of cut. Addition of more aircraft broadens the range of effective angles from which to attack the runway. The method used for developing the response surfaces should be scaled down to a single aircraft to determine if the results differ for a single attack as opposed to a strike package. The intuitive answer is that the results should be similar.

2. Continuation of Dr. Whitehead's (55) work might prove worthwhile in attempting to trade off probability of closure versus the maximum amount of damage inflicted on the runway surface. These goals tend to conflict. The former is an indication of whether the field will be closed for any period of time, while the latter gives an idea of how long the field could be expected to remain closed.

3. More work is needed on the feasibility of fitting response surface curves. If families of curves can be developed, then aircrews could look up their particular attack conditions to determine the best attack heading and weapons spacing.

4. In the present version of AAPMOD, the only method of damaging structures is by craters. The portion of building upon which the crater infringes is considered destroyed. This ignores both fragmentation and blast damage, which will be significant factors in structural damage.

5. In attacking an airfield, there is probably a "best" way to attack and a surrounding region of "good" ways to attack. It is possible that the best attack from an effectiveness standpoint may not be the best attack from a survivability perspective. The most likely avenues of attack may be the best defended. Identification of regions containing near-optimal points would be useful for analysis of tradeoffs between survivability and effectiveness -- the ultimate goal of any airfield attack study.

6. The non-linear transformations and non-linear optimization techniques may streamline the search for optimal airfield attack parameters. Hillier and Lieberman (25:748) suggest non-linear programming using Kuhn-Tucker conditions when considering non-linear problems. The difficulties of obtaining valid response surface fits indicate possible non-linearities in the airfield attack surface.

Conclusion

This thesis effort has succeeded in its primary objectives. Extensive modification was performed to the AAPMOD input procedures. The old program was divided into three sections which now permit easy management of target, weapon, and attack pattern databases. Previous to this effort, no such capability existed for microcomputers, and mainframe computer programs were unwieldy. Now the entire AAPMOD package can be executed on microcomputers common throughout the tactical fighter community.

Verification and validation of AAPMOD were performed during the course of the analysis. Previous coding errors were discovered and corrected. Extensive testing of random number generators was

accomplished. Output from all AFIT versions of AAPMOD was cross-checked and found to produce nearly identical results. Output based on the same number stream produced identical results from the PASCAL microcomputer version and the FORTRAN mainframe version of AAPMOD.

Several statistical experiments were performed. A screening experiment was employed to identify significant airfield attack parameters. Probability of arrival, weapons type, delivery error, number of weapons, weapon reliability, attack heading, and weapon spacing were all found to be significant. Response surface experimentation successfully fitted a regression equation to airfield attack responses in the form of probability of runway cut and expected number of craters to be filled to re-open the airfield. Stationary points identified from the regression equations lay outside feasible regions, but results could be bounded to produce usable results.

While response surfaces can be fitted to the airfield attack problem, excessive computer runs would be required. Regions of interest necessarily would be very small to obtain a valid response surface fit. Extrapolation outside the immediate area of fit was shown to be highly inaccurate. Restricted ranges of airfield attack input parameters would not fulfill aircrew needs for planning attacks. In addition, any change in the target airfield would render the response surface invalid.

This analysis recommends use of on-the-spot computer simulation for airfield attack planning. Databases for target airfields, weapons, and attack patterns can be maintained easily with newly available microcomputer versions of AAPMOD and associated programs. Airfield attack plans can be quickly programmed and analyzed at wing and squadron levels.

XI. BIBLIOGRAPHY

1. Anderson, Kenneth C. and Ronald B. Nenner. A Wild Weasel Penetration Model. Unpublished MS Thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, 1982.

2. Arnett, Capt. J.M. Toward Validation of Computer Simulation Models In Operational Test and Evaluation. Unpublished MS Thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, 1979.

3. Banks, Jerry and John S. Carson, II. Discrete-Event System Simulation. Englewood Cliffs NJ: Prentice-Hall, Inc., 1984.

4. Battilega, John A. and Judith K. Grange. The Military Applications of Modeling. Foreword by J.S. Przemieniecki. Dayton OH: AFIT Press, 1984.

5. Box, G.E.P. and K.B. Wilson. "On the Experimental Attainment of Optimum Conditions," Journal of the Royal Statistical Society, Ser. B, 13:1, (1951).

6. Breuer, D. Wallace. The Fundamentals of Weapons Engineering, Volumes I & II. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, February 1984.

7. Callero, Monti et al. Toward an Expert Aid for Tactical Air Reporting. Report on use of knowledge engineering in tactical planning problems. Defense Advanced Research Projects Agency, Arlington VA, January 1981.

8. Cochran, William G. and Gertrude M. Cox. Experimental Designs (Second Edition). New York: John Wiley & Sons, Inc., 1957.

9. Colony, Stephen F. Readiness Officer. Telephone interview. 2750th Civil Engineering Squadron, Wright-Patterson AFB OH, 8 January 1985.

10. David, H.A. Order Statistics. New York: John Wiley and Sons, Inc., 1970.

11. Davies, Owen L. The Design and Analysis of Industrial Experiments. New York: Hafner Publishing Company, 1963.

12. Department of the Air Force. Flight Manual, A-10A. T.O. 1A-10A-1. Farmingdale NY: Fairchild Republic Company, 15 December 1981.

13. Department of the Air Force. Functions and Basic Doctrine of the United States Air Force. AFM 1-1. Washington: HQ USAF, February 1979.

14. Department of the Air Force. Minimum Operating Strip Selection. Student Handout. Prime Base Engineering Emergency Force (Prime BEEF) Training, Air Force Engineering and Services Center, Eglin AFB FL, 1983.

15. Douglas, Katherine H. Evaluation of Optimum Conditions for Attacking A Runway. Unpublished MS Thesis. School of Engineering, University of Florida, Gainesville FL, 1982.

16. Emerson, D.E. AIDA: An Airbase Damage Assessment Model. Report for the Director of Planning, Programming, and Analysis, HQ USAF. The Rand Corporation, Santa Monica CA, September 1976.

17. Flynn, Brian. "Numerical Accuracy of Least-Squares Computer Programs," ACCESS, 13-20, (Jul/Aug 1983).

18. Foley, Michael J. and Stephen G. Gress, Jr. A Simulation Model to Evaluate Aircraft Survivability and Target Damage During Offensive Counterair Operations. Unpublished MS Thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, March 1984.

19. Golden, Major August. Radar Electronic Warfare. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, May 1983.

20. Goodson, Winfred L. USAFE/XO, Consultation with Guest Lecturer, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 21 April 1983.

21. Goodson, Winfred L. Simulation of Terminal Engagements. Sabre Tiger (Alpha). Volume I. Summary. (U) , Assistant Chief of Staff, Studies and Analysis, (Air Force), Washington, 20 February 1970.

22. Hachida, Howard M. A Computer Model to Aid the Planning of Runway Attacks. Unpublished MS Thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base OH, December 1982.

23. Hicks, Charles R. Fundamental Concepts in the Design of Experiments. New York: Holt, Rinehart and Winston, 1982.

24. Hill, W.J. and W.G. Hunter. "A Review of Response Surface Methodology," Technometrics, 8:4, (Nov 1966).

25. Hillier, Frederick S. and Gerald J. Lieberman. Introduction to Operations Research (Third Edition). San Francisco: Holden-Day, Inc., 1980.

26. Hoeber, Francis P. Military Applications of Modeling. Ex Libris Maj J. Coakley. Department of Operational Sciences, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH.

27. Joint Technical Coordinating Group for Munitions Effectiveness. Attack Assessment Program (AAP), Volume 1 -- User's Manual. 61JTCG/ME-80-3-1. Washington: 1 Jun 80.

28. Joint Technical Coordinating Group to Munitions Effectiveness. Joint Munitions Effectiveness Manuals. (U). Classified. Washington: 15 September 1976.

29. Kizer, Gary G. and Donald W. Neal. A Simulation Model to Evaluate Close Air Support Kill-to-Loss Ratios. Unpublished MS Thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1983.

30. Knuth, Donald E. The Art of Computer Programming, Volume II. Reading MA: Addison-Wesley Publishing Company, 1981.

31. Law, Averill M. and W. David Kelton. Simulation Modeling and Analysis. New York: McGraw-Hill Book Company, 1982.

32. Leek, Warren J. and Richard W. Schmitt. Survivability Study of a FLIR Equipped Fighter on a Night Penetration of a Soviet Army. Unpublished MS Thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, March 1981.

33. Lewis, T.G. and B.J. Smith. Computer Principles of Modeling and Simulation. Boston: Houghton Mifflin Company, 1979.

34. Meyer, Deborah G. and Benjamin F. Schemmer. Interview with General Wilbur L. Creech. Armed Forces Journal. (January, 1983).

35. Miglin, Robert N. "AAPMOD," An Interactive Computer Model for Analysis of Conventional Weapons Effectiveness. Unpublished MS Thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1984.

36. Miller, John M. "Low Angle Low Drag." USAF Fighter Weapons Review, 26-34 (Summer 1981).

37. Mirham, C.A. Simulation: Statistical Foundations and Methodology, New York: Academic Press, 1972.

38. Montgomery, Douglas C. Design and Analysis of Experiments (Second Edition). New York: John Wiley & Sons, Inc., 1984.

39. Morierity, Thomas. Chief Analyst, Studies and Analysis, Headquarters United States Air Force, Washington, DC. Class Lecture. February 1984.

40. Myers, Raymond H. Response Surface Methodology, Virginia Polytechnic Institute, 1976.

41. Office of the Comptroller General. Report to the Congress of the United States -- Models, Data, and War: A Critique of the Foundation for Defense Analyses. Office of the Comptroller General, 12 March 1980.

42. Peck, Allen G. Validating the Runway Attack Methodology of the Naval War College Air Model. Unpublished MS Thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1985.

43. Pemberton, John C. A Generalized Computer Model for the Targeting of Conventional Weapons to Destroy a Runway. Unpublished MS Thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1980.

44. Pritsker, A. Alan B. and Claude Dennis Pegden. Introduction to Simulation and SLAM. New York: John Wiley and Sons, 1979.

45. Publications and Graphics Division. FORTRAN Version 5 Common Library Mathematical Routines Reference Manual, 60483100, Revision B. Control Data Corporation, Sunnyvale CA, 8 September 1979.

46. Quattromoni, Anthony F. Catalog of Wargaming and Military Simulation Models. Studies, Analysis, and Gaming Agency; Organization of the Joint Chiefs of Staff, Washington DC, May 1982.

47. Schemmer, Benjamin F. "NATO's New Strategy: Defend Forward, But Strike Deep." Armed Forces Journal. (November 1983).

48. Schmidt, J.W. and R.E. Taylor, Simulation and Analysis of Industrial Systems, Homewood IL: Richard D. Irwin, Inc., 1970.

49. Shannon, Robert E. Systems Simulations: The Art and Science, Englewood Cliffs NJ: Prentice-Hall, Inc., 1975.

50. Supreme Headquarters Allied Powers Europe. NATO Study of the Balance of Military Power. Belgium: May 1982.

51. Thierauf, Robert J. and Richard A. Grosse. Decision Making Through Operations Research. New York: Wiley and Sons, Inc., 1972.

52. -----. Turbo Pascal Reference Manual. Borland International, Inc., Scotts Valley CA, 1983.

53. USAF Fighter Weapons School. Instructional Text Weapons Delivery F-111, Vol I. Text for course F-111 OIDOAI/DOWI, Tactical Air Command, Nellis AFB NV, September 1981.

54. Wikner, Dr. N.F. (Fred). "Interdicting Fixed Targets with Conventional Weapons." Armed Forces Journal. (March, 1983).

55. Whitehead, Dr. James M. Telephone interview. Discussion of optimum attack heading considering both probability of cut and number of craters produced. 10 January 1985.

56. Whitney, Charles A. "Generating and Testing Pseudorandom Numbers," Byte. Vol 9, No 11 (October 1984).

VITA

Captain Thomas King Green was born on 9 March 1953 in Ruislip, England. Upon graduation from Harrison High School, Harrison, New York in 1970, he entered the United States Air Force Academy and graduated with a Bachelor of Science in Computer Science in June 1974.

A Senior Pilot, Captain Green has a broad background of tactical experience gained from flying 1300 hours in F-4 Phantom II aircraft. He was a distinguished graduate of his pilot training class and earned his wings in July 1975. He then became an RF-4C pilot at the 62nd Tactical Reconnaissance Squadron, Shaw Air Force Base, South Carolina. In August 1978, Captain Green transitioned to the F-4E aircraft and was assigned to the 90th Tactical Fighter Squadron, Clark Air Base, Republic of the Philippines, where he flew the air superiority mission in the F-4E and the wild weasel mission in the F-4E and F-4G. In February 1981, he began a tour of duty at the 10th Tactical Reconnaissance Wing, RAF Alconbury, England, where he served as the Chief of Maintenance Quality Assurance, Maintenance Supervisor of the 10th Aircraft Generation Squadron, and chief test pilot.

Captain Green's academic background includes graduation from the Air University's Squadron Officer School and the Air Command and Staff College. He attended the University of Southern California at the Clark Air Base Study Center and graduated with a Master of Science in Systems Management in January 1981. Captain Green entered the School of Engineering, Air Force Institute of Technology, in August 1983.

Permanent Address: 214 Key Haven Road

Key West, Florida 33040

VITA

Major David Alan Roodhouse was born on 20 November 1950 in Hannibal, Missouri. He graduated from Port Huron High School in 1969. He entered the United States Air Force Academy and, in 1973, became a distinguished graduate with Bachelor of Science Degrees in Physics and Computer Science.

A Senior Pilot, Major Roodhouse has extensive air-to-ground experience with over 2000 fighter hours in both A-7 and A-10 aircraft. He began his tactical career by flying A-7's at Myrtle Beach Air Force Base, South Carolina in 1975. When the 356th Tactical Fighter Squadron transitioned from A-7's to A-10's, he was the first lieutenant in the United States Air Force to fly the A-10 aircraft. In 1979, Major Roodhouse was transferred to the 81st Tactical Fighter Wing, RAF Bentwaters, England, as part of the initial A-10 cadre in the first operational A-10 squadron in Europe. During this period he attended the USAF Fighter Weapons School at Nellis Air Force Base, Nevada. In 1981 he was assigned to instructor duties at the 355th Tactical Training Wing at Davis-Monthan Air Force Base, Arizona.

Major Roodhouse was a distinguished graduate from the Air University's Squadron Officer School in 1978 and completed Air Command and Staff College in 1981. He earned a Masters of Business Administration through the Golden Gate University in 1983. Major Roodhouse entered the School of Engineering, Air Force Institute of Technology, in August 1983.

Permanent Address: Route 1, Box 780
Lake Waccamaw, North Carolina 28450

APPENDIX A

USER MANUAL FOR THE ATTACK ASSESSMENT PROGRAM PACKAGE

Table of Contents

I. Introduction

This appendix comprises the user manuals for the Attack Assessment Program (AAP) package, consisting of four programs:

1. Attack Assessment Program--MODIFIED (AAPMOD).
2. AAP Target database generator program (AAPTGT).
3. AAP Weapons and Attack Pattern database generator program (AAPWPN).
4. AAP Mission database generator program (AAPMSN).

This appendix addresses system requirements for running these programs, compilation of PASCAL source code into executable microcomputer programs, and detailed instructions describing how to use each program.

II. Program Summaries

AAPMOD

AAPMOD is used to calculate probabilities of target kill based on the input data contained in a single input database file. The only user interface required is the specification of the input and output file names. The program reads the input file, calculates the output statistics, and stores the results in the output file. Depending on the input database and the computer system on which the program is run, the program may take from a few seconds to as long as several hours, or even days, to complete.

AAPTGT, AAPWPN, and AAPMSN

AAPTGT, AAPWPN, and AAPMSN are fully interactive programs which allow the user to build a new database or modify an existing database for input to AAPMOD.

AAPTGT. AAPTGT handles the target database. A new database can be generated, or an existing one can be modified. The target complex is defined here. Up to 3 runways, 30 pavements and a total of 112 targets may be entered into the database. Runways and taxiways are categorized as pavements, and all other targets are considered to be "non-pavements." Targets must be assigned a hardness code ranging from 1 to 11, which is then used to indicate how much damage a particular weapon will inflict on the target when hit. Targets also are assigned a group number ranging from 1 to 15 so that output data will be grouped as desired. The amount of repairs that can be made to the pavements and the priority of these repairs are loaded into the database. When the

program ends, the new version of the file is written on the disk and is ready for use by program AAPMSN.

AAPWPN. AAPWPN handles the weapons and attack pattern database. A new database can be generated, or an existing one can be modified. The weapons to be used and the attack patterns to be flown are defined here. Weapons are defined by the sizes of craters they produce upon impact with targets of the different hardness codes specified in the target database. Up to 6 different types of weapons may be defined in this manner. Once the weapons are defined, the program next loads the pattern data. Up to 12 different attack patterns may be defined. For each pattern, the number of weapons dropped, weapons reliability, range errors, and deflection errors must be specified. When the program ends, the new version of the file is written on the disk and is ready for use by program AAPMSN.

AAPMSN. AAPMSN combines the databases from files written with AAPTGT and AAPWPN with additional data input during execution of AAPMSN and writes a single file ready to be used by AAPMOD. The additional data include a random number seed, number of iterations desired, frequency of reports during the iterations, the level of significance desired for the analysis and its corresponding Standard Normal test statistic, and target attack data. The attack data include the target to be attacked, the number of passes over the target, aimpoints, attack direction for each pass, reattack information, and enroute attrition. After the program ends, AAPMOD may be run using the newly created disk file.

III. System Requirements

Hardware

The database generator programs (AAPTGT, AAPWPN, and AAPMSN) require a microcomputer with 64 kilobytes (K) of random access memory and 1 disk drive (or equivalent). The AAPMOD program requires a microcomputer with a minimum of 192K of random access memory and 1 disk drive (or equivalent). All of the microcomputer programs were designed for interactive display on a video screen with a minimum of 80 columns and 24 lines.

It is highly recommended that the microcomputer be equipped with an 8087 math coprocessor for reasonable AAPMOD execution times. Alternatively, AAPMOD may be run on a mainframe computer. Hardware to transfer files from the microcomputer to the mainframe computer may be desired.

Contingent on the scenario and the machine, AAPMOD execution time can be quite lengthy. The NEC Advanced Personal Computer (NEC APC) with a floating point (Intel 8087) coprocessor unit ran sample test cases in approximately 3 to 8 minutes. These same runs required about 3 to 6 seconds on the Control Data Corporation (CDC) 6600 CYBER. Without the 8087 coprocessor, a 16-bit machine with a 5 MegaHertz 8086 central processor unit will require about 7 times as long to run the same program based on results from a NEC APC.

Software

The microcomputer AAP series is written in Borland International's TURBO PASCAL and is specific to this version of PASCAL. TURBO PASCAL is readily available from Borland International, 4113 Scotts Valley Drive,

Scotts Valley, CA 95066 at very reasonable cost. Versions are available for the following operating systems: Digital Research's CP/M-80, CP/M-86, and Concurrent CP/M-86; Microsoft's MS-DOS; and International Business Machine's PC-DOS.

To run the series, command files produced by TURBO PASCAL for the microcomputer are recommended. These command files must have been generated for the specific microcomputer terminal to be used. If the microcomputer is equipped with an 8087 math coprocessor, command files produced by the TURBO-87 version of TURBO PASCAL will be needed to make use of the coprocessor. Should these command files be unavailable, the standard, non-8087 files will run, but at the slower speed of the same computer not equipped with the 8087 coprocessor.

Alternatively, a version of TURBO PASCAL compatible with the microcomputer may be used to compile and generate the executable command files. A version of TURBO PASCAL (TURBO-87) would be required to compile command files compatible with the 8087 coprocessor, if the microcomputer is so equipped. In order to compile the programs, the PASCAL source code files for each program must be available while executing TURBO PASCAL.

The operating system must accept filenames in the form xxxxxxxx.xxx, where xxxxxxxx and xxx are a series of alphanumeric characters (A through Z and the digits 0 through 9) separated by a period. Though the programs will run with different operating systems, the operating system must be compatible with TURBO PASCAL. Furthermore, different systems require different amounts of available memory; consequently, minimum configuration systems such as 8-bit, 64K machines

may not be able to run the AAP programs unless there remains a sufficient transient program area to fit them.

AAPTGT, AAPWPN, and AAPMSN were tested successfully with TURBO PASCAL Versions 1.0, 2.0, and 2.1. The programs also tested successfully on the following systems: a 64K Kaypro 4 running Digital Research's CP/M-80, Version 2.2; a 56K Electronic Control Technology TT-10 system running CP/M-80, Version 2.2 with ZCPR2; a 64K Integrand Single Board Computer (S-100 Super-Quad) running CP/M-80, Version 2.2; and a 256K NEC APC running Digital Research's CP/M-86, Version 1.107, Concurrent CP/M-86, Version 2.00, and Microsoft's MS-DOS, Version 2.11. Files generated by the programs will be completely compatible among machines, operating systems, and versions of TURBO PASCAL.

AAPMOD was tested successfully on a NEC APC with 256K, running the operating systems mentioned in the previous paragraph for the NEC APC and all previously mentioned versions of TURBO PASCAL.

Program AAPMSN produces a disk file that is ready for input to AAPMOD. This input file is compatible with the TURBO PASCAL version and the CYBER CDC 6600 FORTRAN 5 version of AAPMOD. Therefore, by transferring the disk file to the CYBER computer, AAPMOD may be executed in FORTRAN 5 with very quick execution times. Software for transferring files to the CYBER may be desired. To run the mainframe version on a computer other than the CYBER might require minor modification to the FORTRAN 5 source code in order to make it compatible with a different FORTRAN compiler.

IV. Using TURBO PASCAL to Generate AAP Programs

The most convenient way to run the AAP series of programs is to obtain or create command files from the PASCAL source code. This section of the user manual describes how to create these files, and assumes that the user has:

1. A microcomputer meeting the minimum hardware specifications given on page A-6.
2. Borland International's TURBO PASCAL (any version).
3. An operating system compatible with TURBO PASCAL.
4. PASCAL source code for the AAP programs to be compiled on a disk compatible with the microcomputer being used.

The source code for the four programs is in the form AAPxxx.PAS and AAPxxx1.PAS, where AAPxxx is either AAPMOD, AAPTGT, AAPWPN or AAPMSN, and AAPxxx1 is AAPMOD1, AAPTGT1, AAPWPN1, or AAPMSN1. Each of the files AAPxxx.PAS contains a single statement, as follows: {$IAAPxxx1.PAS}. This statement is the TURBO PASCAL compiler directive to INCLUDE the file named AAPxxx1.PAS. Since these files are most likely too large for the TURBO PASCAL editor to load, the INCLUDE directive is necessary to allow command file creation. For other methods of compilation and file editing, refer to the TURBO PASCAL Reference Manual.

How to Create a Command File

To create a command file, only 2 source code files need be present on the disk: the pair of AAPxxx.PAS and AAPxxx1.PAS files (for example, AAPMOD.PAS and AAPMOD1.PAS to create the command file AAPMOD.COM or AAPMOD.CMD). First, execute the TURBO PASCAL program. If the microcomputer has an 8087 coprocessor and an 8087 version of TURBO

PASCAL is available, be sure to execute the TURBO-87 program instead of TURBO. Respond "N" (no) to the question "Include error messages (Y/N)?" if system memory is a limitation (64K systems), since this saves approximately 1.5K additional memory while TURBO PASCAL is executing. From the main TURBO menu, first "log in" the disk drive containing the diskette of AAPxxx files. To do this, type the letter: L. TURBO prompts with "New drive:" and you should respond: D <RETURN>, where D is the disk drive designator (probably A or B). Now slowly (to allow different menus to stabilize on the video display) type the following sequence of letters: OCQC. Then, when asked to enter the work file name, type AAPxxx and press <RETURN>. TURBO PASCAL now should compile the program and write the command file to disk. When complete, indicated by compilation statistics displayed on the video screen and a return of the TURBO PASCAL prompt (the > character), type the letter Q to exit TURBO. Unless errors occurred, the command file should be ready to run by typing the command AAPxxx.

Possible Errors

TURBO may issue an error indicating memory overflow. If this occurs, be sure the work file was the one with the INCLUDE directive in it, and not the one containing the large source code. If the work file was specified correctly, then the microcomputer configuration does not have sufficient memory to allow successful compilation. This may occur with one or more programs. The only solution is to use a different configuration, such as a microcomputer with more memory or a smaller operating system. This error should not occur with 256 K (or larger) machines. Another error may be caused by a disk with insufficient room

for the command file to be written, resulting in a disk full error. If this occurs, files should be copied to a disk with more space available; the only files that need to be on the new disk are the pair of AAPxxx.PAS and AAPxxx1.PAS files to be compiled. If an error message indicating the disk is "READ-ONLY" occurs, the disk may have a write-protect tab missing (8 inch disks) or present (5 1/4 inch disks), or the operating system may have to be reset (for example, using control-C for CP/M systems). For these or any other errors, refer to the Operating System or TURBO PASCAL Reference Manuals.

V. Attack Assessment Program--MODIFIED (AAPMOD)

AAPMOD is a program that may be executed on a mainframe computer in FORTRAN 5 or on a microcomputer in TURBO PASCAL. See the hardware and software specifications (page A-6) for descriptions of system requirements. For either system, AAPMOD requires a data input file and room for an output file to be created and written. The input file may be created with the AAP series of database generator programs (described elsewhere in this appendix) on a microcomputer. The file also may be created or modified with any system editor, as long as the input format is correct. This method of input file creation is not recommended unless the user is intimately familiar with the structure of the files and how AAPMOD uses them. The input file format is compatible with either the mainframe or the microcomputer versions of AAPMOD; hence a file existing on one system may be transferred to the other and successfully executed.

Mainframe execution speeds generally will be much faster than the microcomputer; however, the results will be equally valid from either system. For a microcomputer with an 8087 math coprocessor and compatible TURBO PASCAL software, the execution speed will be tolerable, taking about the same amount of central processor unit (CPU) minutes to complete as the CYBER CDC 6600 takes in CPU seconds. Without the 8087 coprocessor, the execution will take longer by a factor of about 7 times that of the 8087-equipped system CPU time.

To run AAPMOD, first compile the source code if an executable file is not available. (For TURBO PASCAL, refer to compilation instructions, page A-9.) Then be sure the input file is available (loaded on disk,

or in an accessible mainframe computer file). Next, command AAPMOD, and follow the directions on the screen. When asked for the input file, type the filename and press <RETURN>. Then provide the name of a new, unused file for the program output, and press <RETURN>. After both names have been entered, AAPMOD will perform airfield attack assessment and write the results onto the output file specified. No further action is required until the program ends. Then, to see the results, print the output file on 132-column paper, or view the output file on a 132-column video screen.

VI. Attack Assessment Program Target Database Generator (AAPTGT)

This program, designed for a microcomputer, creates the target database to be used in conjunction with programs AAPWPN and AAPMSN in order to create an input file for AAPMOD. There are two modes of execution: new file creation, or modification of an existing file. When creating a new file, AAPTGT requires the entire target complex to be defined. When modifying an existing database, targets may be added, inserted, or deleted, and other data also may be changed.

To begin the program, an executable version must exist on the disk. If there is no executable file (AAPTGT.COM or AAPTGT.CMD), one must be created by compiling the PASCAL source code. Refer to the discussion earlier in this appendix for compilation instructions. Assuming the executable file is available, type the command AAPTGT and press <RETURN> to begin.

Creating a New Target Database

First you will have the option of selecting the execution mode of creating a new target database or modifying an existing one. Select the appropriate choice from the menu and press <RETURN>. The following discussion is based on creating a new database; however, for modification, the same guidelines apply.

Next you will be asked if you would like a review of the input prompts and options. Until you are familiar with them, answer yes (type y and press <RETURN>). When the review is complete, follow the instructions on the screen requesting you provide the filename of the target database file.

The first screen of the review of input options and prompts begins a description of the target input options and what they mean. There are two ways to input targets in this program: Target center and end-midpoint coordinates, and these will be described later. The units of measurement may be expressed in feet, meters, or any other units you desire, but once you have chosen a unit, stay with the same unit throughout the remainder of the programs. The coordinate system is a positive right-hand, rectangular coordinate system, with an origin of your choice. A useful origin is the center of the main runway, since it makes the runway definition simple. Then, target locations are entered by their X and Y coordinates with reference to the chosen origin. The axis is defined by the positive X-axis as zero degrees. The axis measurement increases from 0 to 360 degrees as you move counter-clockwise from the X-axis. When entering the targets, their size must be specified. Be sure to specify the smaller dimension as the width, and the longer one as the length.

For target center coordinates, specify the center of the target in X-Y coordinates with respect to the origin. For end-midpoint coordinates, you enter the midpoint coordinates of the shorter dimension (width) ends of the target. For a runway, for example, you would enter the midpoints of the runway ends. It is important to enter the midpoints in the correct sequence. The program prompts for the X and Y coordinates of the leftmost short dimension. In this case, leftmost assumes you are viewing the target complex with the positive X-axis pointing to the right, and therefore, the positive Y-axis pointing straight up. The program also asks for the X and Y coordinates of the opposite ends. In case there are no leftmost ends, then the

X-coordinates of both ends are the same and X1 = X2. In this case, when entering the Y-coordinates, enter the smallest (bottom-most) value first. Based on end-midpoint entries, AAPTGT calculates the equivalent X-Y coordinates, axis orientation, and target length. You must input the target width.

If the length you enter or AAPTGT calculates is smaller than the width you specify, the program will display an error message and repeat the input sequence for target location and size. Once the length and width information is correct, AAPTGT will check pavement targets to determine if the width is larger than 899 (feet, meters, or whatever units you are using). If larger than 899, a message may appear on the screen indicating a flag is set. This flag will be set upon the first occurrence of a pavement width greater than 899 and will remain unchanged as long as any pavement in the database is wider than 899. The flag enables or suppresses AAPMOD execution of routine OVLAP. This routine searches for overlapping areas of craters and adjusts the damaged area accordingly. For wide targets, the routine takes excessively long to execute, and therefore is suppressed by setting the flag. You have no control over this flag in program AAPTGT, but you may override the flag when building the final input file for AAPMOD using program AAPMSN.

AAPTGT treats each target as one of 2 types: a pavement or a building (non-pavement). A pavement may be a runway or a taxiway. A building is any other kind of target. To each target, a hardness code must be assigned. The range of hardness codes that may be assigned is from 1 to 11. You must determine how many different hardness codes you want to assign before entering target data. A hardness code determines

what level of damage a target will sustain from a particular weapon type. For example, a bomb would cause a larger crater in a soft asphalt runway than in a hardened reinforced concrete runway and perhaps more damage to the taxiways than either of the runways. If your target complex had these two runways and taxiway surfaces of uniform hardness, you would want 3 different hardness codes for the pavements and additional hardness codes for other targets as desired.

Each target must be assigned a group identifier, ranging from 1 to 15. This will not affect the results in terms of damage sustained, but serves to group statistics in a more convenient way for the output of results. For example, you may want to group each runway separately, all taxiways together, and all aircraft shelters together. The output will calculate individual reports of expected hits per target and also will give expected hits per target group.

For the targets, you may specify up to 3 runways, 30 pavement targets (runways plus taxiways), and 112 total targets (runways, taxiways, and buildings). You must enter runways first, followed by taxiways, and then all remaining targets. If you do not follow this sequence, the program will reject your attempts to enter incorrect data and display appropriate error messages guiding you to correct the mistake. If you find you cannot enter a target because of the incorrect sequence, you can save the file and edit it later in the modification mode, or start over from the beginning.

When entering runways, you must specify the minimum clear length and width required for takeoff and landing operations. AAPMOD uses this information to determine if a runway is damaged enough to prevent takeoffs and landings based on the distribution of craters. It will

search the damaged runway surface for an undamaged portion corresponding to the specified minimum width and length to determine runway kill.

If a major taxiway could be used for TOL operations and you want to calculate results accordingly, enter the taxiway as one of the 3 permitted TOL surfaces. Otherwise, a taxiway is entered as a pavement with a minimum clear length for takeoff and landing operations of zero. In this case, the minimum width specified for taxi operations will be used to determine if a taxiway is killed. The path for taxiing will not necessarily lie in a straight line, since taxiing aircraft may maneuver around craters (unlike aircraft taking off or landing). Note that AAPMOD will still evaluate runways that are killed (in terms of takeoff and landing operations) to see whether they may be used for taxiing to other runways that are not yet killed.

All other targets are called buildings, or non-pavements. They must be assigned separate hardness codes from the pavement targets, even if the same level of damage would be inflicted by a weapon.

After each target is entered, you are asked if you want to quit. Answering no causes the program to prompt for the next target input. Answering yes causes the program to move to the next set of questions concerning repair capability. First you are asked how many patches of pavements resources will allow. Then you are asked to provide the priority for repairing the pavements. The repair information is needed for analysis of multiple attacks, but not in this particular implementation of AAPMOD. While the choice of specific numbers is completely arbitrary, you must provide entries for the database to be completed successfully.

Following these last two questions, the target database is complete. AAPTGT then creates the new target file, and writes the data to disk. Then the program ends.

Modifying a Target Database

You can use AAPTGT to modify existing target bases by selecting the appropriate response from the menu when first executing the program. The following discussion pertains to the modification mode of program AAPTGT. If you are unfamiliar with the structure of the target database, you should review the procedures describing how to create a new database. Subsequent discussion assumes you have some working knowledge of the prompts and inputs described in the previous section of this program description. When you begin program AAPTGT and select the modification mode, you may review these prompts and inputs in the same manner as the creation mode.

The file you want to modify must be available on a diskette, but does not have to be on the same one as program AAPTGT. You may specify the disk drive when you enter the filename. AAPTGT will attempt to read the file you specify. If an error message similar to "I/O error 10, PC=1234, Program aborted" appears on the screen, there is a format error in the input file or you did not specify the correct file. In either case, AAPTGT will not be able to process that particular file.

In the modification mode, the current value of the minimum width for taxi is shown, and you may update it if desired. Then the hardness codes are displayed, and you may change these, too. Next you may choose from 5 options on a menu for editing the target element data: you may delete a target of your choice, add a new target to the end of the list,

insert targets into the list, review the target matrix, and finish editing. The insert mode is especially useful for inserting pavements in the correct sequence. Remember, the sequence of the target database must be runways first (a maximum of 3), followed by taxiways (a maximum of 30 minus the number of runways), and all other targets last (maximum of 112 targets total).

The options for editing a target database are presented for your choice from an on-screen menu. The following are descriptions of the various options. For specific instructions on how to enter data, please refer to the previous section, "Creating a New Target Database."

Delete a Target. To preclude deleting a target mistakenly, the program will accept your request to delete a target and ask that you confirm the target number you entered. Once you confirm the deletion, the target will be removed, and all other targets with a higher position number in the matrix will be shifted to fill in the hole created by the deletion. If the target you delete happens to be the last pavement with a width greater than 899, a message will appear on the screen advising you that a flag is reset. This is the flag that enables or suppresses AAPMOD execution of routine OVLAP, described in the database creation mode section. The program will advise whether the target in fact was deleted or not, and you also may confirm this by a review of the target matrix when the program returns you to the menu. If you delete all of the targets, the program automatically places you in the "Add a target" mode, since you must define at least one target in order to run programs AAPMSN and AAPMOD.

Add a Target. You may add targets to the end of the target matrix as long as there are less than 112 targets currently defined; targets

may be deleted to make room for a subsequent addition, if necessary. Adding a target means that the new target definition will be added to the end of the matrix as it currently exists. For example, if there are 4 targets in the matrix, adding a new target will cause the new target to be placed in position number 5. If 112 targets already are defined, the program will issue an error message and return you to the menu. Other error messages will be displayed if you attempt to enter a target out of the correct sequence, define more than 3 runways, or enter more than 30 pavements. Otherwise, you will be able to load the new target definition in the same manner as in the creation mode of the program. After the target is entered, the program will return you to the menu.

Insert a Target. You may insert a target anywhere in the target matrix as long as there are less than 112 targets currently defined; targets may be deleted to make room for a subsequent insertion, if necessary. Inserting a target means that the new target definition will be inserted into the table at the position you specify, displacing the target in that position. The displaced target, along with targets following the displaced target, are shifted up in position number to make room for the new target. For example, if there are 5 targets, and you insert at position 2, target # 1 will remain unchanged in position, the new target will become target # 2, and old targets # 2 through 5 will become targets # 3 through 6. If you try to insert a target at the end of the target matrix where no target currently exists, the program automatically adds the target, effectively changing the option to "Add a target." If 112 targets already are defined, the program will issue an error message and return you to the menu. Other error messages will be displayed if you attempt to insert a target out of the correct sequence,

define more than 3 runways, or enter more than 30 pavements. Otherwise, you will be able to load the new target definition in the same manner as in the creation mode of the program. After the target is entered, the program will return you to the menu.

Review the Target Matrix. When you select this option, the contents of the target matrix will be displayed on the screen, up to 10 targets at a time. To continue the display, simply press any key. Note the abbreviations under TGT TYPE (target type): TOL is a takeoff and landing pavement, TWY is a taxiway pavement, and BLDG is a building or non-pavement target. After all targets have been displayed, the program returns you to the menu.

Finish & Save the New Database. When you are finished with the review and modification of the target matrix, enter this choice from the menu. You will then have an opportunity to change the crater repair data. Finally, the program will create the new file and save the old file. The original (old) version of the file named FILENAME.XXX will be renamed FILENAME.BAK as a backup. The new version will have the same name as the original file before editing. At this point, the program ends. If subsequently you should want to edit the backup file, you first will have to rename it with some other name, since the AAP programs will not accept a file with the extension .BAK. Refer to the Operating System Manual for the computer you are using for instructions on how to rename the file (most likely the REN command).

VII. Attack Assessment Program Weapons and Attack Pattern Database Generator (AAPWPN)

This program, designed for a microcomputer, creates the weapon and attack pattern database to be used in conjunction with programs AAPTGT and AAPMSN in order to create an input file for AAPMOD. There are two modes of execution: new file creation, or modification of an existing file. When creating a new file, AAPWPN requires all weapons and attack pattern information to be defined. When modifying an existing database, data may be added, inserted, or deleted.

To begin the program, an executable version must exist on the disk. If there is no executable file (AAPWPN.COM or AAPWPN.CMD), one must be created by compiling the PASCAL source code. Refer to the discussion earlier in this appendix for compilation instructions. Assuming the executable file is available, type the command AAPWPN and press <RETURN> to begin.

Creating a New Weapons and Attack Pattern Database

First you will have the option of selecting the execution mode: creating a new target database or modifying an existing one. Select the appropriate choice from the menu and press <RETURN>. Then follow the instructions on the screen requesting you provide the filename of the weapons and attack pattern database file. The following discussion is based on creating a new database; however for modification, similar guidelines apply.

At this point in the program, you will begin building the database. The following describes information you will need to enter. Much of

this information is summarized and displayed on the video screen when you execute program AAPWPN.

The program is split into 2 major sections: building the weapons database and building the attack pattern database. The first section builds the weapons database. AAPMOD assesses damage based on sizes of craters caused by weapons that hit targets specified in the target database. AAPMOD uses an array, called the crater table, to store this crater size information. The first part of program AAPWPN guides you while you build this array.

The Weapons Database. The 3 dimensions (i, j, and k) of the crater table array comprising the weapons database are as follows:

i: Hardness Code, Range 1 to 11. This is a defined level of target hardness, thickness, type of material, and similar factors all combined into one, categorical hardness code. For each hardness code you defined in the target database, you must enter crater sizes corresponding to the warhead types and possible interactions described below.

When creating a new database, you must enter the hardness code information corresponding to a specific existing target database. If you do not know this information, execute program AAPTGT in the modification mode and specify the existing target database filename. Follow through program AAPTGT until the hardness information is displayed, record this information for input to AAPWPN, and abort program AAPTGT by pressing control-C (hold down the <CONTROL> key while simultaneously depressing the letter C). Then execute program AAPWPN.

AAPWPN prompts you for the total number of hardness codes entered in program AAPTGT, then the number of hardness codes

corresponding to pavements. A summary of this information is displayed on the screen, and it should match the same display given in program AAPTGT. If this information is incorrect, you may reenter the data until the display is correct. Be sure the hardness information is accurate when creating a new weapons database, because you cannot correct this particular error later in the modification mode.

j: A Defined Type of Warhead, Range 1 to 6. Example: Mk-82 500 pound general-purpose bomb is one possible type of warhead. An AGM-65B Maverick Missile is another possible warhead. A minimum of 1 up to a maximum of 6 different crater size sets may be defined. Note that AAPMOD will not compute crater sizes; you load this information here, using program AAPWPN. If you wish to load different crater information for one particular kind of warhead under different delivery conditions, you must consider each set of data as a separately defined warhead. For example, you may want to define crater sizes for the main runway at Base X: j = 1, Mk-82 bomb delivery from an F-4 Phantom at 500 knots, 20 degrees of dive and, for j = 2, delivery from an A-10 Thunderbolt II at 325 knots, 10 degrees of dive.

When creating a new database, you are asked to enter the number of different types of warheads. Enter the number of warheads you wish to define for this database. Note that you cannot later define attack patterns for which there are no defined weapons. Therefore, be sure to define all weapons you will need for building the attack pattern database in section 2 of the program.

k: Interaction 1 or Interaction 2 (k = 1 and k = 2). There are actually 4 interactions: Interactions 1 and 2 for targets defined as pavements, and interactions 1 and 2 for targets defined as

non-pavements (buildings). Only one set of interactions is entered into the array, and this is determined by the hardness code corresponding to a pavement or non-pavement target. AAPWPN automatically prompts for the correct interaction, but it is important to recognize what the different combinations represent. Since all interactions involve crater sizes, the following first describes how AAPMOD handles craters and then discusses the different interactions in detail.

AAPMOD evaluates target damage by comparing the locations of targets to the locations and crater sizes of weapon impacts. However, AAPMOD uses square craters for its computations, because this makes the algorithm for evaluation of pavement kill status more efficient. Miglin, in his thesis on the FORTRAN 5 version of AAPMOD, shows how use of square craters is a good approximation, where errors would tend to cancel each other over the course of the program. Because of AAPMOD's use of square craters, you have a choice of how you will enter crater size data: square craters or circular craters. If you enter square craters, enter one half of the length of the side of the square. If you enter circular craters, enter the radius; AAPWPN will calculate the area of the circular crater and convert this to an equivalent area square crater for the database. Important: Remember to use the same units of dimension (feet, meters, etc) you used in defining the target database.

Pavement Interactions. If the target is a pavement (taxiway or runway), the important piece of information is the damaged pavement surface area which effectively denies use of that portion of surface for aircraft use. When requesting you to load weapons interaction data, AAPWPN starts with the pavement codes (hardness codes associated with pavements). This information is entered as follows:

Interaction 1. Interaction 1 corresponds to the crater size which would disrupt takeoff and landing operations. This includes the physical size of the crater hole and the additional pavement surface around the hole where any cracks, buckling, and rubble would preclude high speed takeoff and landing operations. AAPWPN calls this interaction "Deny-TOL size." If the pavement code is associated with a taxiway only, the "Deny-TOL size" would not be used; however, subsequent modification of the target database may need the same hardness code for a takeoff surface. Therefore, you should provide a correct crater size for takeoff and landing considerations.

Interaction 2. Interaction 2 corresponds to the crater size which would disrupt taxi operations. This includes the physical size of the crater hole, but little additional pavement, since aircraft can taxi slowly over surfaces with minor cracks, buckles, and deposits of rubble. Note that for pavements, the crater size for interaction 1 normally will be larger than that of interaction 2. AAPWPN calls this interaction "Deny-taxi size." Since all pavements are considered taxi-capable, correct data must be entered for all of the pavement hardness codes.

Buildings (Non-Pavements). After all of the pavement hardness codes have been processed, AAPWPN requests data for the hardness codes associated with buildings (non-pavements). If the target is a building, the important piece of information is the damaged structure area which effectively denies use of that portion of the structure. This information is entered as follows:

Interaction 1. Interaction 1 corresponds to the crater size which would result from a target near-miss. In other words, if the

bomb missed the structure, but landed nearby, this is the size of the crater that would result. AAPWPN calls this interaction "Near-miss size."

Interaction 2. Interaction 2 corresponds to the crater size which would result from a direct hit on the target. Note that for non-pavements, the crater size for interaction 1 normally will be smaller than that of interaction 2. AAPWPN calls this interaction "Direct hit size."

The Attack Pattern Database. The second major section of program AAPWPN guides you while you build the attack pattern database. This portion of the program begins by asking you how many patterns you wish to define. You must define at least 1 pattern and may define up to 12.

Next a menu is presented on the screen, giving a choice of general purpose weapons, cluster bomblet units (CBU), or guided munitions. There are 2 choices for CBU based on the footprint of bomblets that will occur: rectangular, with bomblets impacting uniformly over the rectangular area and no voids in coverage; or elliptical, with bomblets uniformly distributed over the elliptical area except that voids in coverage are possible. You will have to specify the dimensions of the footprint and additionally, for the elliptical CBU footprint, the size of the voids.

A weapons pattern can be considered as the end result of a weapons delivery pass producing a pattern of weapon impacts on the ground. For each pattern definition, you must specify how many weapons will be delivered per pass (from 1 to 12 weapons). Exception: for guided weapons, the program defaults to 1 weapon per pass.

Next you must enter the reliability (probability of functioning) of the weapon/cannister fuze, taking into consideration all factors such as proper fuze settings, arming, cockpit switch settings, weapons release parameters, and weapon impact angle. This should not include the reliability of individual bomblets inside the cannister of a cluster munition; you will enter this information later, if applicable.

The program prompts for the weapon's crater table index. Enter the warhead index corresponding to the crater table dimension j discussed in the first section of the AAPWPN program description. In other words, of the weapons already defined in the database, identify which weapon is being employed.

The next set of data is based on which one of the weapons descriptions you chose from the menu. For unguided munitions (general purpose and CBU), you must enter the delivery and ballistic range and deflection errors probable (REP and DEP), using the same dimensional units (feet, meters, etc) as the crater sizes and target coordinate system. Range errors lie along the aircraft's ground track while deflection errors are referenced perpendicular to the ground track. The concept of REP is that 50% of the range errors will fall within the range error probable distance, and similarly for DEP, that 50% of the deflection errors will fall within the deflection error probable distance. For circular error probable (CEP), CEP is merely a simplifying assumption that REP and DEP are the same. Enter the CEP value for both REP and DEP if only CEP is known.

Ballistic errors, also known as ballistic dispersion, include the random errors induced by slight differences in drag, stability, mass, bomb rack ejection cartridges, and other factors for a particular weapon

type. The end result of ballistic error is an impact displacement from the actual point at which the weapon is aimed (the aimpoint based on an error-free pipper). The delivery error includes avionics error, aircrew error in releasing the weapon at the wrong time, weather, and similar factors, but does not include target misidentification (aiming error). This is an important distinction; AAPMOD will stochastically determine aiming, delivery, and ballistic errors as follows: The aiming error is based on a triangular distribution built into the design of the model. The delivery and ballistic errors are based on bivariate normal distributions with the parameters you will enter into the attack pattern database.

For CBU munitions, further information is required. First you must specify how many bomblets each CBU cannister contains. Second, give the reliability of the individual bomblets, expressed as a probability of proper bomblet fuze functioning. Third, input the footprint length and width. The last CBU data is for elliptical footprints only: the void length and width, which create a doughnut shape of the bomblet area of ground coverage. Remember to be consistent with the dimensional units when specifying length and width.

For guided munitions, you may choose the form of input for errors as REP/DEP or standard deviation (sigma). REP and DEP are as described previously for unguided munitions. Recall that if you wish to enter a CEP, enter figures based on CEP = REP = DEP. Standard deviation, also known as sigma, allows you to input the errors based on the normal distribution data for the weapon, if available. Instead of the delivery and ballistic errors for unguided munitions, you will enter guidance errors for guided munitions. There are 3 categories of guidance which

you must consider: optimal, near-miss, and gross-error. Specify the optimal errors based on optimum weapon performance (good weather, properly functioning avionics, and weapon release parameters well within tolerance). Specify the near-miss errors based on degraded conditions (marginal weather, minor avionics malfunctions, or weapons release on the edge of the acceptable weapons envelope). Specify the gross-error data based on a major guidance malfunction, such as when a missile goes ballistic or a smart weapon loses the designator signal.

The last inputs for guided munitions are probabilities of guidance. Enter the probability that the weapon will encounter optimum guidance conditions. Then enter the cumulative probability that the weapon will encounter near-miss or optimum guidance conditions. This cumulative probability will be the same as [1 - Pr(gross-error conditions)], and will always be equal to or greater than the probability of optimum guidance conditions alone.

AAPWPN continues to loop through the same set of questions for each of the patterns to be defined. After the last pattern has been defined, the file is written to disk and saved with the file name specified at the start of program execution. Then the program ends.

Modifying an Existing Database

The following discussion pertains to the modification mode of program AAPWPN. If you are unfamiliar with the structure of the weapons and attack pattern databases, you should review the procedures describing how to create a new database. Subsequent discussion assumes you have some working knowledge of the prompts and inputs described in the previous section of this appendix.

The file you want to modify must be available on a diskette, but does not have to be on the same one as program AAPWPN. You may specify the disk drive when you enter the filename. AAPWPN will attempt to read the file you specify. If an error message similar to "I/O error 10, PC=1234, Program aborted" appears on the screen, there is a format error in the input file or you did not specify the correct file. In either case, AAPWPN will not be able to process that particular file.

After AAPWPN successfully reads the existing database you wish to modify, the program begins with a display of the target hardness categories. There is no way to change this data; the information is presented for your review. If you discover an error in the hardness codes, you will have to correct the problem by executing program AAPWPN in the database creation mode. Note that target hardness information is based directly on a target file. Weapons must be defined with reference to known targets, and the entire crater table is designed with the hardness codes as one of the dimensions (i) of the 3-dimension crater table (i, j, k).

The modification mode, like the creation mode, has two program sections: modifying the weapons database and modifying the attack pattern database. Anytime during the program you may abort the editing session, thus discarding any current changes, but saving the input file completely unchanged (including the filename). To abort an editing session, press control-C (hold the <CONTROL> key depressed while simultaneously typing the letter C). If you run the modification mode until the program terminates normally, your newly edited version of the database will be stored in the disk file with the same name as the old file you originally specified. Additionally, the old version of the

database also will be saved with a filename in the form FILENAME.BAK to help in case of an editing error.

Throughout the program you will have numerous chances to review the weapons and pattern information, presented in a compact, self-documenting display on the video screen. You should frequently review these weapons and patterns matrices to be sure the changes you make are reflected accurately. Once you are familiar with their design, you will gain valuable insight into the structure of the database and become increasingly proficient at designing your model.

The Weapons Section. The weapons section begins with a menu of 7 choices. You may delete, add, insert or redefine a weapons description; review the weapons and attack pattern matrices; or exit the weapons section and move on to the attack pattern section. Once you leave the weapons section, you may not return without ending the program and restarting. Following are descriptions of the menu choices.

Delete a Weapon and its Corresponding Patterns. This is the most complicated of the weapons modification processes, because the existing pattern matrix is based on the defined weapons in the crater table. If you delete a weapon which has been referenced by a pattern, that pattern is no longer valid, and so it, too, will be deleted automatically. Note that if you delete all of the weapons, you also will deplete the pattern matrix. To preclude your accidental deletion of weapons and patterns, the program will accept your request to delete a weapon and print pattern information on the screen. If any patterns are listed, they are subject to deletion along with the weapon; however, you will be given a chance to review the pattern matrix to be sure of your decision, then asked to confirm the deletion. The program then

will advise whether the weapon was in fact deleted or not, and you also may confirm this by a review of the matrices when the program returns you to the menu.

Another factor involved in deleting weapons from the crater table: the pattern matrix contains references, by position, to the weapons in the crater table. If you delete a weapon, all other weapons with a higher position number in the table will be shifted to fill in the hole created by the deletion. AAPWPN automatically adjusts the pattern matrix references so that the remaining patterns still agree with the weapons for which they were defined.

It is possible to delete all of the weapons and patterns from the database. In this case, the program automatically places you in the "Add a weapon" mode, since patterns cannot be defined without first defining weapons. If this situation occurs accidentally, you should abort the program by pressing control-C, and your original database file will not be disturbed. Then restart AAPWPN.

Add a Weapon. You may add weapons to the end of the crater table as long as there are less than 6 weapons currently defined; weapons may be deleted to make room for a subsequent addition, if necessary. Adding a weapon means that the new weapon definition will be added to the end of the table as it currently exists. For example, if there are 4 weapons in the table, adding a new weapon will cause the new weapon to be placed in position number 5. If 6 weapons already are defined, the program will issue an error message and return you to the weapons menu. Otherwise, you will be able to load the new weapon definition in the same manner as in the creation mode of the program. After the weapon is entered, the program will return you to the menu.

Insert a Weapon. You may insert a weapon anywhere in the crater table as long as there are less than 6 weapons currently defined; weapons may be deleted to make room for a subsequent insertion, if necessary. Inserting a weapon means that the new weapon definition will be inserted into the table at the position you specify, displacing the weapon in that position. The displaced weapon, along with the weapons following the displaced weapon, are shifted up in position number to make room for the new weapon. For example, if there are 5 weapons, and you insert at position 2, weapon # 1 will remain unchanged in position, the new weapon will become weapon # 2, and old weapons # 2 through 5 will become weapons # 3 through 6. If you try to insert a weapon at the end of the crater table where no weapon currently exists, the program automatically adds the weapon, effectively changing the option to "Add a weapon." If 6 weapons already are defined, the program will issue an error message and return you to the menu. Otherwise, you will be able to load the new weapon definition in the same manner as in the creation mode of the program. After the weapon is entered, the program will return you to the menu.

Another factor involved in inserting weapons into the crater table: the pattern matrix contains references, by position, to the weapons in the crater table. If you insert a weapon, all other weapons with a higher position number in the table will be shifted as described in the previous paragraph. AAPWPN automatically adjusts the pattern matrix references so that the remaining patterns still agree with the weapons for which they were defined.

Redefine a Weapon. The purpose of redefining a weapon is to allow the weapon to be redefined without altering the pattern matrix

reference to the weapon position number. For example, if a pattern existed for weapon 2, currently defined as a Mk-82 500 pound general purpose bomb, you could retain the same pattern definition but redefine the weapon to model Mk-84 2,000 pound bombs using this option of the program. Use caution with this feature: if the pattern is built for weapon # 2, a cluster bomblet unit (CBU) bomblet, and you redefine weapon # 2 to be a Mk-82, your pattern still will be designed for CBU, except that the bomblet crater sizes will be much bigger (Mk-82 craters) than previously defined. You will be able to load the new weapon definition in the same manner as in the creation mode of the program. After the weapon is entered, the program will return you to the menu.

Review the Weapon Matrix. When you select this option, the crater table format will be shown on the first screen, and after you press any key to continue the display, the entire crater table will be shown on the second screen. Then the program will return you to the menu.

Review the Attack Pattern Matrix. When you select this option, the attack pattern information will be displayed on the screen. Because there is so much information pertaining to each pattern, only one pattern is shown per screen. The display pauses until you press any key to continue, then moves on to the next pattern. After the last pattern is shown, the program will return you to the menu.

End Weapon Section Editing. When you are finished with the review and modification of the weapons matrix, enter this choice from the menu. The program will move to the attack pattern section for further review and editing.

The Attack Pattern Section. This portion of the program allows you to edit the pattern database. The modes of editing are similar to those in the weapons section. The main difference is that there are no internal interactions between the pattern matrix and the weapons matrix; whatever changes you make to the patterns, the weapons crater table will remain unchanged. The attack pattern section begins with a menu of 6 choices. You may delete, add, or insert patterns; review the weapons or pattern matrices; or save the edited version of the database and end the program. Following are descriptions of the menu choices.

Delete a Pattern. To preclude deleting a pattern mistakenly, the program will accept your request to delete a pattern and ask that you confirm the pattern number you entered. Once you confirm the deletion, the pattern will be removed, and all other patterns with a higher position number in the matrix will be shifted to fill in the hole created by the deletion. The program will then advise whether the pattern was in fact deleted or not, and you also may confirm this by a review of the pattern matrix when the program returns you to the menu. If you delete all of the patterns, the program automatically places you in the "Add a pattern" mode, since you must define at least one pattern in order to run programs AAPMSN and AAPMOD.

Add a Pattern. You may add patterns to the end of the pattern matrix as long as there are less than 12 patterns currently defined; patterns may be deleted to make room for a subsequent addition, if necessary. Adding a pattern means that the new pattern definition will be added to the end of the matrix as it currently exists. For example, if there are 4 patterns in the matrix, adding a new pattern will cause the new pattern to be placed in position number 5. If 12 patterns

already are defined, the program will issue an error message and return you to the menu. Otherwise, you will be able to load the new pattern definition in the same manner as in the creation mode of the program. After the pattern is entered, the program will return you to the menu.

Insert a pattern. You may insert a pattern anywhere in the pattern matrix as long as there are less than 12 patterns currently defined; patterns may be deleted to make room for a subsequent insertion, if necessary. Inserting a pattern means that the new pattern definition will be inserted into the table at the position you specify, displacing the pattern in that position. The displaced pattern, along with the patterns following the displaced pattern, are shifted up in position number to make room for the new pattern. For example, if there are 5 patterns, and you insert at position 2, pattern # 1 will remain unchanged in position, the new pattern will become pattern # 2, and old patterns # 2 through 5 will become patterns # 3 through 6. If you try to insert a pattern at the end of the pattern matrix where no pattern currently exists, the program automatically adds the pattern, effectively changing the option to "Add a pattern." If 12 patterns already are defined, the program will issue an error message and return you to the menu. Otherwise, you will be able to load the new pattern definition in the same manner as in the creation mode of the program. After the pattern is entered, the program will return you to the menu.

Review the Weapon Matrix. When you select this option, the crater table format will be shown on the first screen, and after you press any key to continue the display, the entire crater table will be shown on the second screen. Then the program will return you to the menu.

Review the Attack Pattern Matrix. When you select this option, the attack pattern information will be displayed on the screen. Because there is so much information pertaining to each pattern, only one pattern is shown per screen. The display pauses until you press any key to continue, then moves on to the next pattern. After the last pattern is shown, the program will return you to the menu.

Save Data and End the Program. When you are finished with the review and modification of the pattern matrix, enter this choice from the menu. The program then will save the newly edited version of the database in the disk file with the same name as the old file you originally specified. Additionally, the old version of the database will be saved with a filename in the form FILENAME.BAK in case you need the old file in its original form.

VIII. Attack Assessment Program Mission Database Generator (AAPMSN)

This program, designed for a microcomputer, combines a target database and a weapons and attack pattern database with additional information entered during AAPMSN program execution. The combined data are written onto a single disk file in a format compatible with the input requirements of AAPMOD. Therefore, program AAPMOD may be executed immediately with the new file produced by successful completion of program AAPMSN.

Program AAPMSN has one mode of execution: new file creation. To begin the program, an executable version of AAPMSN must exist on the disk. If there is no executable file (AAPMSN.COM or AAPMSN.CMD), one must be created by compiling the PASCAL source code. Refer to page A-9 for compilation instructions. Additionally, you must have the 2 existing database files (a target database and a weapons and attack pattern database) available on disk and be able to identify the files when prompted for their names by AAPMSN. Assuming the required files are available, type the command AAPMSN and press <RETURN> to begin.

How to Create the AAPMOD Input File

The start of program AAPMSN asks you for the file names of the target database file, the weapons and attack pattern database file, and the output file to be generated by AAPMSN. The program then will attempt to read the 2 input files. If an error message similar to "I/O error 10, PC=1234, Program aborted" appears on the screen, there is a format error in one of the input files or you did not specify the correct files required by AAPMSN. A different type of error may be generated by AAPMSN, advising you of a discrepancy between the 2 input

files in the number of hardness levels defined. AAPMSN reads both input files, checks the hardness levels, and prints this error message if the hardness levels are not the same. The descriptions of programs AAPTGT and AAPWPN discuss how the number of hardness codes for pavement targets and non-pavement targets must be defined. If you want to review the hardness levels or any other information in the input database files, use the modification mode of program AAPTGT for the target file or AAPWPN for the weapons and attack pattern file.

As soon as the input files are read successfully, AAPMSN allows you to review the target, weapons, and attack pattern matrices. You will be able to review these at other times throughout the program, too. Then the program prompts you for initial mission analysis information.

Building the Mission Package. For the next series of questions, the program has a set of default values. If you press <RETURN> in response to the questions, the default values will be loaded automatically. At the end of these questions, you will have a chance to review the values and change them if desired. The inputs, discussed in the next paragraph, have default values as follows: the flag that enables or suppresses AAPMOD execution of routine OVLAP remains unchanged, random number seed = 987654567, number of samples = 200, results to be reported once, level of significance = 0.05, and Z = 1.645.

The first message you will see after reviewing the database matrices is a description of the flag that enables or suppresses AAPMOD execution of routine OVLAP. The message will indicate the status of this flag based on the particular target database you are using. If you built the target file with program AAPTGT, the flag is set according to

the width of the widest pavement. A width greater than 899 units (feet, meters, or whatever the target database used) will set the flag to suppress AAPMOD routine OVLAP. If all of the targets are less than this size, the flag will not be set, and AAPMOD routine OVLAP will be enabled. You have the option of leaving the flag at its current setting or overriding the flag. You may want to override the flag and suppress routine OVLAP to save execution time. Routine OVLAP searches for overlapping craters and adjusts the total area damaged by the amount of overlap. When the routine is disabled, this search will not be accomplished for any of the target pavements, and execution time will be reduced by an order of magnitude.

AAPMSN asks you if you want to specify a random number seed. If you answer no, the program will load its own default seed, 987654567. Note that the seed is not used by the TURBO PASCAL version of AAPMOD; the seed is loaded in case of input to the mainframe version of AAPMOD which does require a seed.

Next you must enter the number of Monte Carlo samples you want to run, where each sample represents a complete mission profile. AAPMOD will produce output statistics for any specified number of samples, but more samples result in more confidence in the answer. More samples also take more time to compute. You must decide the acceptable tradeoff between accuracy and execution time. This can be determined by use of a confidence interval.

Start with 200 samples and generate the AAPMOD output statistics. Decide which variable is of most interest to you, and develop the confidence interval for it. If the expected number of hits on the target receiving the most hits is of interest, you are in luck: AAPMOD

automatically calculates what is called the confidence half-interval for 90% and 99% confidence. For instance, suppose that target element #1 has an expected number of hits numbering 14 with a 90% confidence half-interval of 0.5 and a 99% confidence half-interval of 1.0. This means that 99 out of 100 runs of the simulation will give a number of hits on target element #1 numbering 14 ± 1.0. This is probably quite acceptable.

If other statistics are more important, as probably will be the case, you can calculate your own confidence half-interval. The confidence half-interval is based on the Student's t-statistic. For more than 30 samples, the t-statistic can be approximated by the normal statistic. For the normal statistic, the required numbers to use are $Z_{\alpha/2} = 1.645$ for the 90% confidence half-interval and $Z_{\alpha/2} = 2.576$ for the 99% confidence half-interval.

Suppose that for 200 samples, the probability of cut for target element #1 (a runway) is 0.500 with a SIGMA of 0.035. Then the 90% confidence half-interval is:

$$90\%\ CI = (Z_{\alpha/2})\ (\text{SIGMA})\ /\ (\text{number of samples})^{\frac{1}{2}}$$
$$= (1.645)\ (0.035)\ /\ (200)^{\frac{1}{2}} = 0.0041$$

Using similar calculations, the 99% confidence half-interval is 0.0064. In this case, 99 out of 100 runs would be expected to result in probabilities of cut in the range 0.500 ± 0.0064. This is probably accurate enough for any envisioned use of AAPMOD. In fact, the number of samples probably could be reduced. Remember, running AAPMOD with less than 30 samples will require use of the t-statistic for confidence interval calculations: you would need to find these values in

statistical tables. The easiest solution is to use 200 samples and check the accuracy afterwards.

Next, enter the desired reporting interval. If you are only interested in the final results of the run, enter the same figure as the number of samples. If you specified more than 200 samples for the program to run, the next prompt will ask you if you want AAPMSN to optimize its sampling procedures. Answering yes causes the program to run 200 samples, then compute how many additional samples it will run in order to produce the level of significance you specify. The next input is the level of significance for the analysis. The level of significance is the same as 1 minus the confidence level. Typical values are 0.10 (90% confidence), 0.05 (95% confidence), 0.025 (97.5% confidence), 0.01 (99% confidence), 0.005 (99.5% confidence), and 0.001 (99.9% confidence). A high confidence level means that you are reasonably sure that the results are accurate. The lower the level of significance, the more samples will be required, and hence longer computer execution times will result. A commonly used level of significance is 0.05. Following this entry, you must provide the "Z" or Standard Normal test statistic for the level of significance. Typical "Z" values are given on the screen for your convenience; however, you may use the Standard Normal tables to obtain the correct value for other levels of significance.

AAPMSN continues with a partial recap of the mission package data entered so far. Review this information carefully, because you may correct any errors now, but not later. If there are errors, the program will repeat the first section questions for you to reenter. If there

are no errors, the program continues with the final portion of the mission: entering the flight line-up.

Building the Flight Line-Up. The flight line-up is the sequence of aircraft attack passes to be flown over the target complex. AAPMOD allows a maximum of 32 passes flown by a maximum of 32 aircraft. Each aircraft will fly a minimum of 1 and a maximum of 2 passes over the target complex. Therefore, a wave of 32 passes may be flown by a minimum of 16 aircraft assigned 2 passes each, a maximum of 32 aircraft assigned 1 pass each, or a combination of aircraft assigned 1 or 2 passes each. When prompted by AAPMSN, enter the total number of passes to be flown. Then enter the number of aircraft participating in the attack. Next, you must describe each pass in detail.

AAPMSN builds the flight line-up by requesting data for each pass in sequence. The program effectively pairs an aircraft number with each pass as follows: Pass number 1 always is paired with aircraft number 1. Pass number 2 will be paired with aircraft number 2 unless you previously assigned aircraft 1 to fly its second pass as pass number 2. Pass number 3 will be paired with the next available aircraft unless you designated aircraft 1 or 2 to fly pass number 3 as its second pass. The process continues until all passes have been defined, or AAPMSN detects that an insufficient number of aircraft are available to complete the pass descriptions. Should this situation occur, AAPMSN displays an error message and starts over again with the first pass definition. Note that if you design the flight line-up correctly, give the correct information (number of passes and number of aircraft), and make no mistakes in designating passes, this error will not occur.

Here is an example flight line-up:

CALLSIGN	ASSIGNED PASSES		ASSIGNED AIRCRAFT
RED	1	2	1
BLUE	3	6	2
GREEN	5	9	4
BLACK	8	10	5
WHITE	4	7	3

In this example, you, the flight leader, brief the 5 flight members, Red, Blue, Green, Black, and White, to fly 2 passes each over the target complex. Red will fly passes 1 and 2, Blue will attack next (pass number 3), White will fly pass number 4, Green will fly pass number 5, then Blue will make his second pass (number 6), and so forth. The aircraft will be assigned by AAPMSN as shown in column 3 above.

For each pass you must enter the element number (from the target matrix) of the target to be attacked. Then enter the preplanned aimpoint using X-Y coordinates based on the coordinate system and units chosen when building the target database. If the pass will involve multiple weapon releases, the preplanned aimpoint should be entered as the midpoint of the stick of weapons. Next enter the attack direction in degrees counter-clockwise from the positive X-axis. For example, if the positive X-axis points east, and the attack is flown in a north-easterly direction (from the south-west of the target complex), you would enter approximately 45 degrees for the attack azimuth. Then you must specify the attack pattern (from the attack pattern database) to be flown this pass.

At this point of the program, a partial recap of the pass definition will be displayed on the screen. You will have a chance to reenter any incorrect information. Once you indicate the information is

correct, you will not have another opportunity to correct it. After the information recap, you must enter the probability that the aircraft survives up to the first pass. AAPMOD uses this probability to determine whether the first pass results in weapons release. Next you will see a display of the current flight line-up. The aircraft numbers displayed are the ones available for the mission. You must identify which pass the current aircraft will fly next. If the aircraft will not fly a second pass, enter 0. From the example on the previous page, if you were describing Green's pass number 2, the correct response to this prompt would be 9. The last entry for the pass definition is the probability that the aircraft makes the second pass. This is the probability that the aircraft is not killed from the time it makes its first pass to the time it expends its weapons on the second pass.

Prior to each pass you may elect to review any of the 3 data matrices (target, weapon, or attack pattern). You also may view the current flight line-up, which could help to prevent incorrect pass assignments. After all of the passes are defined, AAPMSN writes the data onto the disk file you specified at the beginning of the program and displays this filename on the screen. The program then ends, and you may run AAPMOD with the file newly created.

APPENDIX B

PASCAL LISTINGS FOR THE ATTACK ASSESSMENT PROGRAM PACKAGE

This appendix provides the source code listings for the eight PASCAL programs in the microcomputer series of the Attack Assessment Program Package. The programs are written in Borland International's TURBO PASCAL, and are specific to this version of PASCAL. Further details of the hardware and sortware requirements for running these programs and descriptions of the program functions may be found in the User Manual, Appendix A. The programs listed in this appendix are:

Section 1: AAPMOD.PAS and AAPMOD1.PAS

Section 2: AAPTGT.PAS and AAPTGT1.PAS

Section 3: AAPWPN.PAS and AAPWPN1.PAS

Section 4: AAPMSN.PAS and AAPMSN1.PAS

```
{========================================================================}
{================  FILE AAPMOD.PAS    18 Feb 85  ========================}
{========================================================================}

{$IAAPMOD1.PAS  compiler directive to include file AAPMOD1.PAS}
```

```
{===================================================================}
{=============== FILE AAPMOD1.PAS   10 Feb 85 ======================}
{===================================================================}

{                 Airfield Attack Program ( Modified )

  Older version used at Eglin AFB, FL, and 50-60 contractor locations.
  Developed at Oklahoma State University, under contract F08635-79-C-0255,
  for the Joint Technical Coordinating Group for Munitions Effectiveness.

  Modified by Captain Robert N. Miglin, Air Force Institute of Technology,
  March 1984, to provide interactive capability for tactics assessment with
  a program written for the Cyber CDC 6600 in FORTRAN 5.

  Converted from FORTRAN 5 to TURBO PASCAL by Major David A. Roodhouse and
  Captain Thomas K. Green, Air Force Institute of Technology.           }

{===================================================================}
program aapmod ;

label 85,280,340,350,360,370,430,560,580,590,640,650,660,670,720,730,
      760,770,780,790,800,810 ;

const twopi : real = 6.28318530718 ;

var i,ifin,ii,ij,ii3,inflag3,irepr,is,iseed,it,itgtgp,itgttp,itt,
      i3,j,jd,jdef,jj,jk,jr,jrng,jwpntp,k,ka,kj,kk,kk2,km1,kode,kp1,
      kw,kw1,kz,kzp,kzt,kz1,k0,k2,k8,l,lastj,lj,lv,m,mflag,m0,m1,
      mxptch,n,narea,nbom,ncp,nelt,nfill,nflag1,nflag2,nflag3,nmax,nmin,
      npatt,npass,nptrn,nsamp,nsamp1,nsamp2,nsampr,nsampt,ntgps,
      nttt,ntxwy,nvals,nwep,nxtp : integer ;

    apprcw,arfill,arfils,cosp,cost,crazyn,crmax,crmin,cuts,d,dap,diffar,
      error,fill,oecho,passxt,passyt,r,rmaj,rmin,sinp,sint,sumrun,sumstp,
      s1,s2,s3,tbhld1,tbhld2,t1,vmaj,vmin,x,xctr,xiwod,xp,xs1,xs2,x1,x1y1o1,
      x1y1i1,y,yctr,yiwod,yp,ys1,ys2,y1,zalph : real ;

    icrat, i2cut : array[1..4] of integer ;
    icut : array[1..4,1..3] of integer ;
    ihit, ipcut, kh : array[1..3] of integer ;
    ipass : array[1..32,1..2] of integer ;
    ipat : array[1..12,1..4] of integer ;
    ipl : array[1..40] of integer ;
    isav : array[1..800] of integer ;
    itgt : array[1..112,1..3] of integer ;
    lnhits : array[1..112] of integer ;
    npx : array[1..32] of integer ;
    numapr : array[1..4,1..2] of integer ;
```

```
    adm, countr, decar, rapf, rcut, rhit, sigadm, sights, signaf
      : array[1..112] of real ;
    amin, apra, aprmin, arep, astp, dstr, enapfl, sigarp, sigasp, sigcrt,
      snapfl, xc, yc : array[1..3] of real ;
    crit : array[1..112,1..2] of real ;
    crtab : array[1..11,1..6,1..2] of real ;
    gpadac, gpadm, gpadms, gparea, gpht, gphtac, gphts : array[1..15] of real ;
    pass : array[0..32,1..6] of real ;
    patt : array[1..13,1..34] of real ;
    sapr, sapra, sgapr, sgapra, sgcrat, sgmina, smina : array[1..4] of real ;
    save : array[1..800,1..3] of real ;
    sigcts, sigfil : array[1..27] of real ;
    square : array[1..900] of real ;
    stemp : array[1..303] of real ;
    tgt : array[1..112,1..5] of real ;

    lst : text ; { REMOVE THIS LINE AND MODIFY INDAT FOR HARD COPY }

{==================================================================}
{procedure declarations ===========================================}
{==================================================================}
procedure trisub(var rv : real) ;
  var u, x : real ;
begin
  u := random ;
  x := sqrt(2 * u) ;
  rv := 1000 * x - 1000 ;
end ;  {  of procedure trisub  }

{==================================================================}
procedure clstrp ;
  label 10, 25, 41 ;
  var more, quit : boolean ;
      icc, is, istart, it, ix, j, jdp, jm, jr, jz, min : integer ;
      aicc, cstar, swep, temp, tsxu, tsyu, xs, ys : real ;
      isort, jsort : array[1..800] of integer ;
      area : array[1..800] of real ;
begin
  xs := 0 ;
  ys := 0 ;
  tsxu := crit[1,1] ;
  tsyu := crit[1,2] ;
  cstar := 10.0e15 ;
  nmin := n ;
{ ----define area(j) := difficulty of repairing crater j
      changed 28 oct 81 to compute area of square craters---- }
  for j := 1 to n do area[j] := 4 * sqr( crtab[itgt[1,2],isav[k0],1] ) ;
{ ----set up for sweep---- }
25:
  min := 0 ;
  istart := 0 ;
  swep := 10.0e15 ;
```

```
for j := 1 to n do begin
  if ( save[K0 + j - 1,2] + crtab[itgt[1,2],isav[K0 + j - 1],1] ) ys )
                                        and
     ( save[K0 + j - 1,2] - crtab[itgt[1,2],isav[K0 + j - 1],1] ( tsyu )
  then begin
    if min = 0
      then begin
           min := 1 ;
           isort[1] := j ;
           jsort[1] := j ;
         end
      else begin
           it := min ;
           min := min + 1 ;
           quit := false ;
           repeat
             jz := isort[it] ;
             if save[K0 + j - 1,1] + crtab[itgt[1,2],isav[K0 + j - 1],1] (
                save[K0 + jz - 1,1] + crtab[itgt[1,2],isav[K0 + jz - 1],1]
               then begin
                   isort[it + 1] := isort[it] ;
                   it := it - 1 ;
                   if it ) 0
                     then quit := true
                     else isort[1] := j ;
                 end
               else begin
                   quit := true ;
                   isort[it + 1] := j ;
                 end ;
           until quit ;
           it := min - 1 ;
           quit := false ;
           repeat
             jr := jsort[it] ;
             if save[K0 + j - 1,2] + crtab[itgt[1,2],isav[K0 + j - 1],1] (
                save[K0 + jr - 1,2] + crtab[itgt[1,2],isav[K0 + jr - 1],1]
               then begin
                   jsort[it+1] := jsort[it] ;
                   it := it - 1 ;
                   if it ) 0
                     then quit := true
                     else jsort[1] := j ;
                 end
               else begin
                   quit := true ;
                   jsort[it + 1] := j ;
                 end;
           until quit ;
         end ; (  else  )
    end ; (  if statement just below j loop  )
  end ; (  j loop  )
```

```
{ ----execute sweep
     determine difficulty of repairing craters touching frame---- }
10:
  ix := istart + 1 ;
  aicc := 0 ;
  icc := 0 ;
  repeat
    quit := true ;
    if ix <= min then begin
      jm := isort[ix] ;
      if save[k0 + jm - 1,1] - crtab[itgt[1,2],isav[k0 + jm - 1],1] < tsxu
        then begin
              aicc := aicc + area[jm] ;
              icc := icc + 1 ;
              ix := ix + 1 ;
              quit := false ;
            end
          else begin
              if save[k0 + jm - 1,1] - crmax < tsxu then begin
                ix := ix + 1 ;
                quit := false ;
              end ;
            end ;
    end ;
  until quit ;
{ ----compare repair difficulty for frame---- }
  more := true ;
  if cstar > aicc then begin
    cstar := aicc ;
    nmin := icc ;
    xc[1] := xs ;
    yc[1] := ys ;
    if cstar <= 0.0000001 then begin
      xc[1] := xc[1] + crit[1,1] ;
      more := false ;
    end ;
  end ;
{ ------move frame------ }
  if more then begin
    temp := aicc - cstar ;
 41:
    istart := istart + 1 ;
    if istart <= min then begin
      is := isort[istart] ;
      if temp > area[is] then begin
        temp := temp - area[is] ;
        goto 41
      end ;
      if swep > aicc then swep := aicc ;
      tsxu := save[k0 + is - 1,1] + crtab[itgt[1,2],isav[k0 + is - 1],1] +
              crit[1,1] + 0.000000001 ;
```

```
      if tsxu <= tgt[1,4] then begin
        xs := tsxu - crit[1,1] ;
        goto 10 ;
      end ;
    end ;
{ ----sweep finished---- }
    temp := swep - cstar ;
    jdp := 0 ;
    repeat
      quit := true ;
      jdp := jdp + 1 ;
      if jdp > min then begin
        xc[1] := xc[1] + crit[1,1] ;
        more := false ;
      end ;
      is := jsort[jdp] ;
      if temp > area[is] then begin
        temp := temp - area[is] ;
        quit := false ;
      end ;
    until quit ;
    if more then begin
      tsyu := save[k0 + is - 1,2] + crtab[itgt[1,2],isav[k0 + is - 1],1] +
              crit[1,2] + 0.000000001 ;
      if tsyu > tgt[1,5]
        then xc[1] := xc[1] + crit[1,1]
        else begin
            ys := tsyu - crit[1,2] ;
            xs := 0 ;
            tsxu := crit[1,1] ;
            goto 25  ;
          end ;
    end ; {  if more  }
  end ; {  if more  }
end ; {  procedure clstrp  }

{=============================================================================}
procedure mincw( var n, xy : integer; x, y : integer; var lt : integer;
                 var w, ww : real ) ;
{ ----barnett's taxiway program inserted to replace mincw 1 oct 81
      latest version of taxiway  23 april 1982
            nc  :=  max number of craters in a subproblem
            nsub  :=  max number of subproblems to be solved
            n  :=  number of craters in entire problem              ---- }
  type  intarray50 = array[1..50] of integer ;

  label 70, 113, 114, 122, 200, 230, 6000, 7000, 7010, 7020 ;

  var   more, quit, nomore : boolean ;

        i, iflag, isave, iunder, j, jlast, jm, jp, js, k, l, m,
          nc, nf, nfillc, nfm, nl, nnm, np, nsub : integer ;
```

```
      amin, bfeas, bnd, dist, el, eu, rep, rmin, xd, yd : real ;

      istart : array[1..1001] of integer ;

      ibeas, icomp, ipsol, irep, it, list1, list2 : intarray50 ;

      a : array[1..100] of real ;

      wr, wx, wy : array[1..50] of real ;

   procedure check(var inn : intarray50 ) ;

    label 10, 12, 13,17, 20, 500, 501, return ;

    var   jflag, ji, jj, jt, jtemp, jx, jxm : integer ;

          dif, xd, xmin, yd : real ;

    {=============================================================}
    procedure betwn ;

     label 1, 2, 7, 8, 9, 999, 2000 ;

     var   ix, k, kflag, km, kp, k1, k2, l1, l2, nlt, nl1, nl2 : integer ;

           dis, dx, dy, temp, xd, xmax, xmin, yd : real ;

    begin
 { ----(jflag <= 1) implies 'under-over'
      (jflag >= 2) implies 'over-under' ---- }
      kflag := 1 ;
      nl1 := 1 ;
      list1[1] := jx ;
      nlt := 1 ;
      k := jx ;
      xmin := wx[jx] - wr[jx] - crmax - w ;
 { ----construct 'list1' of craters behind jx impinging
      directly or indirectly upon it                    ---- }
 1:
      km := jxm ;
 { ----determine if km impinges upon k---- }
 2:
      if wx[km] >= xmin then begin
        ix := 1 ;
        while (km <> list1[ix]) and (ix <> nl1 ) do ix := ix + 1 ;
        if km <> list1[ix] then begin
          xd := wx[k] - wx[km] ;
          yd := wy[k] - wy[km] ;
          dis := sqrt( sqr(xd) + sqr(yd) ) - wr[km] - wr[k] ;
          if dis < w then begin
            if ( jflag <= 1 ) and ( wy[km] + wr[km] ) ww - w )
                                      or
               ( jflag >= 2 ) and ( wy[km] - wr[km] < w ) then goto 999 ;
```

```
{ --------determine if km impinges upon jxm--------- }
          xd := wx[km] - wx[jxm] ;
          yd := wy[km] - wy[jxm] ;
          dis := sqrt( sqr(xd) + sqr(yd) ) - wr[km] - wr[jxm] ;
          if dis < w then goto 999 ;
          temp := wx[km] - wr[km] - crmax - w ;
          if xmin > temp then xmin := temp ;
          nl1 := nl1 + 1 ;
          list1[nl1] := km ;
        end ;
      end ;
      km := km - 1 ;
      if km > 0 then goto 2 ;
    end ;
    nlt := nlt + 1 ;
    if nlt <= nl1 then begin
      k := list1[nlt] ;
      goto 1 ;
    end ;
{ ----construct 'list2' of craters ahead of jxm impinging
    directly or indirectly upon it                          ---- }
    nl2 := 1 ;
    list2[1] := jxm ;
    nlt := 1 ;
    k := jxm ;
    xmax := wx[k] + wr[k] + crmax + w ;
{ ----determine if kp impinges upon k---- }
7:
    kp := jx ;
8:
    if wx[kp] <= xmax then begin
      for ix := 1 to nl2 do if kp = list2[ix] then goto 9 ;
      xd := wx[k] - wx[kp] ;
      yd := wy[k] - wy[kp] ;
      dis := sqrt( sqr(xd) + sqr(yd) ) - wr[kp] - wr[k] ;
      if dis < w then begin
        if ( jflag <= 1 ) and ( wy[kp] - wr[kp] < w )
                              or
           ( jflag >= 2 ) and ( wy[kp] + wr[kp] > ww - w ) then goto 999 ;
{ --------determine if kp impinges upon jx-------- }
        xd := wx[kp] - wx[jx] ;
        yd := wy[kp] - wy[jx] ;
        dis := sqrt( sqr(xd) + sqr(yd) ) - wr[kp] - wr[jx] ;
        if dis < w then goto 999 ;
        temp := wx[kp] + wr[kp] + crmax + w ;
        if xmax < temp then xmax := temp ;
        nl2 := nl2 + 1 ;
        list2[nl2] := kp ;
      end ;
9:
      kp := kp + 1 ;
      if kp <= jt then goto 8
    end ;
```

```
      nlt := nlt + 1 ;
      if nlt <= nl2 then begin
        k := list2[nlt] ;
        goto 7 ;
      end ;
{ ----determine if list1 impinges upon list2---- }
      for k1 := 1 to nl1 do begin
        l1 := list1[k1] ;
        for k2 := 1 to nl2 do begin
          l2 := list2[k2] ;
          dx := wx[l1] - wx[l2] ;
          dy := wy[l1] - wy[l2] ;
          dis := sqrt( sqr(dx) + sqr(dy) ) - wr[l1] - wr[l2] ;
          if dis < w then goto 999 ;
        end ;
      end ;
      goto 2000 ;
999:
      kflag := 0 ;
2000:
      jflag := kflag ;
    end ; {  procedure betwn  }

  {==========================================================================}
  begin {  procedure check  }
    iflag := 1 ;
    jt := 0 ;
    for jx := 1 to nc do begin
      if inn[jx] < 1 then begin
        jt := jt + 1 ;
        jj := nfm + jx ;
        wx[jt] := save[xy - 1 + jj,x] ;
        wy[jt] := save[xy - 1 + jj,y] ;
        wr[jt] := crtab[lt,isav[xy - 1 + jj],2] ;
      end ;
    end ;
    if jt <= 0 then goto return ;
    it[1] := -1 ;
    if wy[1] - wr[1] >= w then it[1] := 0 ;
    if wy[1] + wr[1] <= ww - w then it[1] := 1 ;
    if it[1] < 0 then begin
      iflag := 0 ;
      goto return ;
    end ;
    jx := 1 ;
10:
    jx := jx + 1 ;
    jxm := jx - 1 ;
    if jx > jt then goto return ;
{ ----can we get over jx?---- }
    if wy[jx] + wr[jx] <= ww - w then begin
      if it[jxm] > 0 then begin
```

```
{ --------do an 'over-over'---------- }
        xmin := wx[jx] - wr[jx] - crmax - w ;
{ --------check back-------- }
        jtemp := jxm ;
13:
        jtemp := jtemp - 1 ;
        if jtemp > 0 then begin
{ ----------does an 'under' impinge upon jx?---------- }
          xd := wx[jx] - wx[jtemp] ;
          yd := wy[jx] - wy[jtemp] ;
          dif := sqrt( sqr(xd) + sqr(yd) ) - wr[jx] - wr[jtemp] ;
          if dif < w then begin
            if it[jtemp] <= 0 then goto 12 ;
          end ;
          if wx[jtemp] >= xmin then goto 13 ;
        end ;
        it[jx] := 1 ;
        goto 10 ;
      end ;
{ ------try for 'under-over'------- }
      jflag := 1 ;
      betwn ;
      if jflag > 0 then begin
        it[jx] := 1 ;
        goto 10 ;
      end ;
    end ;
{ ----can we get under jx ?---- }
12:
    if wy[jx] - wr[jx] < w then begin
      iflag := 0 ;
      goto return ;
    end ;
20:
    if it[jxm] < 1 then begin
{ ------do an 'under-under'------ }
      xmin := wx[jx] - wr[jx] - crmax - w ;
{ ------does an 'over' impinge upon jx ?------ }
      jtemp := jxm ;
17:
      jtemp := jtemp - 1 ;
      if jtemp > 0 then begin
        xd := wx[jx] - wx[jtemp] ;
        yd := wy[jx] - wy[jtemp] ;
        dif := sqrt( sqr(xd) + sqr(yd) ) - wr[jx] - wr[jtemp] ;
        if dif < w then begin
          if it[jtemp] >= 1 then goto 500 ;
        end ;
        if wx[jtemp] >= xmin then goto 17 ;
      end ;
      it[jx] := 0 ;
      goto 10 ;
    end ;
```

```
{ ----try for 'over-under'---- }
    jflag := 2 ;
    betwn ;
    if jflag > 0 then begin
      it[jx] := 0 ;
      goto 10 ;
    end ;
{ ----backtrack---- }
500:
    ji := jx ;
501:
    ji := ji - 1 ;
    if ji >= 1 then begin
      if it[ji] <= 0 then goto 501 ;
      jx := ji ;
      jxm := jx - 1 ;
      if jxm > 0 then begin
        if wy[jx] - wr[jx] >= w then goto 20 ;
        goto 500 ;
      end ;
      if wy[1] - wr[1] >= w then begin
        it[1] := 0 ;
        goto 10 ;
      end ;
    end ;
    iflag := 0 ;
return: .
  end ; { procedure check }
  {====================================================================}

begin { procedure minccw }
  crmax := 0 ;
  if n > 50 then begin
    writeln( lst, '         Number of craters exceeds 50:  n = ',n:5 ) ;
    halt ;(call exit )
  end ;
{ -----changed to compute area of square cra  rs 28 oct 81------- }
  for j := 1 to n do
    if crmax < crtab[lt,isav[xy - 1 + j],2] then begin
      crmax := crtab[lt,isav[xy - 1 + j],2] ;
      a[j] := 4 * sqr( crtab[lt,isav[xy - 1 + j],2] ) ;
    end ;
  nfill := 0 ;
  arfill := 0 ;
{ ----search for subproblems---- }
  istart[1] := 1 ;
  nsub := 1 ;
  nmm := n - 1 ;
  for j := 1 to nmm do begin
    jp := j + 1 ;
    jm := j ;
    el := save[xy + j - 1,x] + crtab[lt,isav[xy - 1 + j],2] ;
    eu := save[xy + jp - 1,x] - crtab[lt,isav[xy - 1 + jp],2] ;
```

```
if el + w <= eu then begin
  repeat
    quit := true ;
    jm := jm - 1 ;
    if jm >= 1 then begin
      if save[xy - 1 + jm,x] + crtab[lt,isav[xy - 1 + jm],2] > el
        then el := save[xy - 1 + jm,x] + crtab[lt,isav[xy - 1 + jm],2] ;
      if save[xy - 1 + jm,x] + crmax > el then quit := false ;
    end ;
  until quit ;
  repeat
    quit := true ;
    jp := jp + 1 ;
    if jp <= n then begin
      if eu > save[xy - 1 + jp,x] - crtab[lt,isav[xy - 1 + jp],2]
        then eu := save[xy - 1 + jp,x] - crtab[lt,isav[xy - 1 + jp],2] ;
        if eu > save[xy - 1 + jp,x] - crmax then quit := false ;
     end ;
  until quit ;
  if el + w <= eu then begin
    nsub := nsub + 1 ;
    if nsub > 1000 then begin
      writeln( lst, '          Subproblems exceed 1000' ) ;

      halt ; (call exit)
    end ;
    istart[nsub] := j + 1 ;
  end ;
 end ;
end ; (  j loop  )
istart[nsub + 1] := n + 1 ;
( ----solve subproblems---- )
for js := 1 to nsub do begin
  nf := istart[js] ;
  nl := istart[js + 1] - 1 ;
  nfm := nf - 1 ;
  crmax := 0 ;
  for j := nf to nl do
    if crmax < crtab[lt,isav[xy - 1 + j],2]
      then crmax := crtab[lt,isav[xy - 1 + j],2] ;
  nc := nl - nfm ;
  if nc > 50 then begin
    writeln( lst, '          Number of craters in subprogram exceeds 50,',
            'nc = ', nc:5 ) ;
    halt ; (call exit)
  end ;
  if nc <= 2 then begin
    bfeas := 0 ;
    np := nf + 1 ;
    if save[xy - 1 + nf,y] + crtab[lt,isav[xy - 1 + nf],2] > w-w then begin
      if save[xy - 1 + nf,y] - crtab[lt,isav[xy - 1 + nf],2] >= w
        then goto 122 ;
      bfeas := bfeas + a[nf] ;
```

```
        nfill := nfill + 1 ;
        irep[nfill] := nf ;
        arfill := arfill + a[nf] ;
        if nc <= 1 then goto 230 ;
        if save[xy - 1 + np,y] + crtab[lt,isav[xy - 1 + np],2] <= ww - w
          then goto 230 ;
        if save[xy - 1 + np,y] - crtab[lt,isav[xy - 1 + np],2] >= w
          then goto 230 ;
        bfeas := bfeas + a[np] ;
        nfill := nfill + 1 ;
        irep[nfill] := np ;
        arfill := arfill + a[np] ;
        goto 230 ;
      end ;
      if nc <= 1 then goto 230 ;
      if save[xy - 1 + np,y] + crtab[lt,isav[xy - 1 + np],2] <= ww - w
        then goto 230 ;
      if save[xy - 1 + np,y] - crtab[lt,isav[xy - 1 + np],2] >= w
        then goto 114 ;
113:
      arfill := arfill + a[np] ;
      bfeas := bfeas + a[np] ;
      nfill := nfill + 1 ;
      irep[nfill] := np ;
      goto 230 ;
114:
      xd := save[xy - 1 + nf,x] - save[xy - 1 + np,x] ;
      yd := save[xy - 1 + nf,y] - save[xy - 1 + np,y] ;
      dist := sqrt( sqr(xd) + sqr(yd) ) -2 * crtab[lt,isav[xy - 1 + np],2] ;
      if dist >= w then goto 230 ;
      if ( save[xy - 1 + nf,y] - crtab[lt,isav[xy - 1 + nf],2] >= w )
                                        and
         ( save[xy - 1 + np,y] - crtab[lt,isav[xy - 1 + np],2] >= w )
        then goto 230 ;
      amin := a[nf] ;
      isave := nf ;
      if a[nf] > a[np] then isave := np ;
      if a[nf] > a[np] then amin := a[np] ;
      arfill := arfill + amin ;
      nfill := nfill + 1 ;
      irep[nfill] := isave ;
      bfeas := bfeas + amin ;
      goto 230 ;
122:
      if nc <= 1 then goto 230 ;
      if save[xy - 1 + np,y] - crtab[lt,isav[xy - 1 + np],2] >= w
        then goto 230 ;
      if save[xy - 1 + np,y] + crtab[lt,isav[xy - 1 + np],2] <= ww-w
        then goto 114 ;
      goto 113 ;
    end;
```

```
{ -----check clear path----- }
   for j := 1 to nc do ipsol[j] := 0 ;
   check(ipsol) ;
   if iflag > 0 then begin
     bfeas := 0 ;
     goto 200 ;
   end ;
{ -----initialization for implicit enumeration----- }
   for k := 1 to nc do begin
     ibeas[k] := 0 ;
     icomp[k] := 1 ;
   end ;
   jlast := 0 ;
   nfillc := 0 ;
   rep := 0 ;
   bfeas := 10.0e20 ;
{ -----forward move----- }
7000:
   jlast := jlast + 1 ;
   iunder := jlast ;
   ipsol[jlast] := 1 ;
   rep := rep +a[nfm + jlast] ;
{ -----test 2----- }
   if rep >= bfeas then goto 7020 ;
{ -----test 1----- }
   check(ipsol) ;
   if iflag <= 0 then goto 7010 ;
   bfeas := rep ;
   for k := 1 to nc do ibeas[k] := ipsol[k] ;
{ -----test 6----- }
7020:
   if nfillc = jlast then goto 70 ;
{ -----backward move----- }
   nfillc := nfillc + iunder - jlast + 1 ;
   ipsol[iunder] := 0 ;
   jlast := iunder ;
   rep := rep - a[nfm + jlast] ;
   if jlast <= 1 then goto 7010 ;
   m := iunder - 1 ;
   nomore := false ;
   k := 0 ;
   repeat
     k := k + 1 ;
     l := iunder - k ;
     if ipsol[l] = 1 then begin
       iunder := iunder - k ;
       nomore := true ;
     end ;
   until ( k = m ) or ( nomore ) ;
```

```
7010:
    if jlast <> nc then begin
      m := jlast + 1 ;
      rmin := 10000 ;
      for k := m to nc do if a[nfm + k] < rmin then rmin := a[nfm + k] ;
    end ;
    bnd := rep + rmin ;
{ -----test 3----- }
    if (bnd >= bfeas) or (jlast = nc) then goto 7020 ;
{ -----test 4----- }
    if ipsol[jlast] = 1 then goto 7000 ;
    for k := 1 to jlast do icomp[k] := ipsol[k] ;
    check(icomp);
{ -----test 5----- }
    if iunder <> jlast then begin
      m := iunder + 1 ;
      for k := m to nc do icomp[k] := 1 ;
    end ;
    if iflag <= 0 then goto 7020 ;
    goto 7000 ;
70:
    arfill := arfill + bfeas ;
200:
    if bfeas > 0
      then for i := 1 to nc do
        if ibeas[i] > 0 then begin
          nfill := nfill + 1 ;
          irep[nfill] := nfm + i ;
        end ;
230:
  end ; {  js loop; do subproblems  }
  cuts := 0 ;
  if nfill <> 0 then cuts := nsub ;
end ; {  procedure mincw  }

{==================================================================}
procedure ovlap( k, lt : integer; x0, y0 : real; itl, itw, kz : integer;
                 var sum : real ) ;

  label 30, 40, 50, 60, 70, 90 ;

  var quitting : boolean ;

      i, j, j6, j6pm, k1, k2, l, l1, l2, l3, m, npi : integer ;

      dxp, dl, sump, x1, x2, x3, y1 : real ;
{ ----initialize---- }
begin
  for i := 1 to itw do square[i] := 0 ;
  sum := 0 ;
  sump := 0 ;
```

```
{ ----find first and last values of x to consider---- }
  l3 := trunc( save[k,1] - crmax + 1 - x0 ) ;
  if l3 < 1 then l3 := 1 ;
  l2 := trunc( save[k - 1 + kz,1] + crmax + 1 - x0 ) ;
  if l2 > itl then l2 := itl ;
  j6 := 1 ;
  m := 0 ;
  l1 := l3 ;
{ ----loop-one square at a time in x
      l := x value at top of square---- }
  l := l1 - 1 ;
  while l < l2 do begin
    l := l + 1 ;
    dxp := 0 ;
    j6 := j6 + m ;
    m := 0 ;
    if j6 > kz
      then l := l2  {  ends procedure  }
      else begin
{ -----if all craters have been considered, return
       loop-crater by crater...consider all craters which
       could possibly intersect in x                          ------ }
        i := j6 - 1 ;
        while i < kz do begin
          i := i + 1 ;
{ ------locate left hand edge of crater------ }
          npi := isav[k - 1 + i] ;
          x1 := save[k - 1 + i,1] - crtab[lt,npi,1] - x0 ;
          if x1 < l - 1 then goto 30 ;
          x2 := save[k - 1 + i,1] - crmax - x0 ;
          if x2 >= l
            then i := kz  {  ends while i < kz loop  }
            else if x1 < l then begin
{ ------left-hand edge of crater lies inside lth square------ }
              dxp := l - x1 ;
              goto 60 ;
{ ------left hand edge of crater is below x-square
        locate right hand edge of crater          ------- }
30:
              x1 := save[k - 1 + i,1] + crtab[lt,npi,1] - x0 ;
              if x1 <= l - 1 then goto 40 ;
              if x1 >= l then goto 50 ;
{ ------right hand edge of crater lies inside lth square------ }
              dxp := x1 - l + 1 ;
              goto 60 ;
{ ------crater i lies entirely left of x-square...no need to consider
        this crater any more                                        ------ }
40:
              x3 := save[k - 1 + i,1] + crmax - x0 ;
              if x3 <= l - 1 then m := m + 1 ;
              goto 90 ;
50:
              dxp := 1 ;
```

```
{ ------crater intersects x-square...check intersections in y------- }
60:
            yi := save[k - 1 + i,2] - crtab[lt,npi,1] - y0 ;
{ ------k1 := index of y-square containing lower edge of crater i------ }
            k1 := trunc(yi) + 1 ;
            if k1 < 1 then k1 := 1 ;
{ ------d1 := % of y-square occupied by crater------ }
            d1 := k1 - yi ;
            if d1 > 1 then d1 := 1 ;
            square[k1] := d1 * dxp + square[k1] ;
            if k1 = itw then goto 90 ;
            k1 := k1 + 1 ;
            y1 := save[k - 1 + i,2] + crtab[lt,npi,1] - y0 ;
            k2 := trunc(y1) ;
            if k2 > itw then k2 := itw ;
            if k2 = itw then goto 70 ;
            d1 := y1 - k2 ;
{ ------load square containing top edge of crater i------ }
            square[k2 + 1] := d1 * dxp + square[k2 + 1] ;
{ ------load intermediate y-squares...d1 := 1------ }
70:
            for j := k1 to k2 do square[j] := square[j] + dxp ;
          end ; {  else if begin  }
90:
       end ; {  while i loop  }
{ ------count squares that are at least half-filled------ }
       for j := 1 to itw do begin
         if square[j] >= 0.5 then sump := sump + 1 ;
         square[j] := 0 ;
       end ;
       sum := sum + sump ;
{ -----if there is a gap in x-values, skip to next x-value needed----- }
       quitting := true ;
       if ( dxp <= 0 ) and ( m <> 0 ) then begin
         j6pm := j6 + m ;
         if j6pm > kz
           then l := 12   {  ends procedure  }
           else begin
               quitting := false ;
               l3 := trunc( save[k - 1 + j6pm,1] - crmax - x0 ) + 1 ;
               if l3 > l
                 then begin
                     l1 := l3 ;
                     l := l1 - 1 ;
                   end
                 else begin
                     l3 := l + 1 ;
                     l1 := l3 ;
                     l := l1 - 1 ;
                   end ;
             end ;
       end ;
```

```
      if quitting then sump := 0 ;
    end ; { else }
  end ; { while loop }
end ; { procedure ovlap }

{==========================================================================}
procedure bldg ;
{ ----assess area remaining undamaged after all hits are
      evaluated for this attack                          ---- }
  var j, ka, npj : integer ;
      dl, dw, d1, d2, oarea, olngth, owdth, ratio, xcen, xh, xoc,
        ycen, yh, yoc  : real ;
begin
  ratio := tgt[l,4] / tgt[l,5] ;
  j := 0 ;
  repeat
    j := j + 1 ;
    dw := sqrt( decar[l] / ratio ) ;
    dl := dw * ratio ;
    xh := ( tgt[l,4] - dl ) / 2 ;
    yh := ( tgt[l,5] - dw ) / 2 ;
    xoc := tgt[l,4] / 2 - xh ;
    yoc := tgt[l,5] / 2 - yh ;
    xcen := save[k0 + j - 1,1] - xh ;
    ycen := save[k0 + j - 1,2] - yh ;
    d1 := abs( ycen - yoc ) ;
    d2 := abs( xcen - xoc ) ;
    npj := isav[k0 + j - 1] ;
    if  ( d1 < ( crtab[itgt[l,2],npj,1] + dw / 2 ) )
                               and
        ( d2 < ( crtab[itgt[l,2],npj,1] + dl / 2 ) )
      then begin
      if ( d1 <= tgt[l,5] / 2 ) and ( d2 <= tgt[l,4] / 2 )
        then ka := 2
        else ka := 1 ;
      owdth := ycen + crtab[itgt[l,2],npj,ka] ;
      if dw < owdth then owdth := dw ;
      if ycen > crtab[itgt[l,2],npj,ka]
        then owdth := owdth - ycen + crtab[itgt[l,2],npj,ka] ;
      olngth := xcen + crtab[itgt[l,2],npj,ka] ;
      if dl < olngth then olngth := dl ;
      if xcen > crtab[itgt[l,2],npj,ka]
        then olngth := olngth - xcen + crtab[itgt[l,2],npj,ka] ;
      oarea := olngth * owdth ;
      decar[l] := decar[l] - oarea ;
      if decar[l] <= 0 then j := n ; { end procedure }
    end ;
  until j = n ;
end ; { procedure bldg }
```

```
{=================================================================}
procedure sort( n, i, xi, yi, zi : integer ) ;
  var it, jo, ko, lo, mo, no : integer ;
      t : real ;
begin
  jo := 0 ;
  i := i - 1 ;
  repeat jo :=  2 * jo + 1  until jo >= n ;
  while jo >= 2 do begin
    jo := trunc(jo / 2) ;
    ko := n - jo ;
    for lo := 1 to ko do begin
      mo := lo ;
      repeat
        no := mo + jo ;
        if save[i + mo,xi] > save[i + no,xi]
          then begin
               t := save[i + mo,xi] ;
               save[i  + mo,xi] := save[i + no,xi] ;
               save[i + no,xi] := t ;
               t := save[i + mo,yi] ;
               save[i + mo,yi] := save[i + no,yi] ;
               save[i + no,yi] := t ;
               t := save[i + mo,zi] ;
               save[i + mo,zi] := save[i + no,zi] ;
               save[i + no,zi] := t ;
               it := isav[i + mo] ;
               isav[i + mo] := isav[i + no] ;
               isav[i + no] := it ;
               mo := mo - jo ;
            end
          else mo := 0 ;
      until mo <= 0 ;
    end ; {  lo loop  }
  end ; {  while  }
end ; {  procedure sort  }

{=================================================================}
procedure noran(var r,sr,d,sd : real) ;
  var a, x : real ;
begin
  x := random ;
  if x = 0 then x := random ;  {  trap to prevent ln(0) next line  }
  a := sqrt( -2 * ln(x) ) ;
  x := random * twopi ;
  r := a * sr * sin(x) ;
  d := a * sd * cos(x) ;
end ;  {  procedure noran  }
```

```
{=====================================================================}
procedure initialize ;  { call after reading input file }
  var i, j : integer ;
begin
  randomize ;  { initializes TURBO PASCAL random number generator }
  nsampr := 1 ;
  nmax := 0 ;
  crmax := 0 ;
  crmin := 1.0e10 ;
  for i := 1 to nelt do begin
    itgttp := itgt[i,2] ;
    for j := 1 to npass do begin
      nptrn := ipass[j,1] ;
      jwpntp := ipat[nptrn,3] ;
      if itgt[i,1] = 1 then begin
        tbhld1 := crtab[itgttp,jwpntp,1] ;
        tbhld2 := crtab[itgttp,jwpntp,2] ;
        if tbhld1 < crmin then crmin := tbhld1 ;
        if tbhld2 < crmin then crmin := tbhld2 ;
        if tbhld1 > crmax then crmax := tbhld1 ;
        if tbhld2 > crmax then crmax := tbhld2 ;
      end ;
    end ;
  end ;
  for i := 1 to nelt do begin
    countr[i] := 0 ;
    sights[i] := 0 ;
    adm[i] := 0 ;
    sigadm[i] := 0 ;
  end ;
  for i := 1 to ntgps do begin
    gphts[i] := 0 ;
    gpadms[i] := 0 ;
  end ;
  for i := 1 to lv + ncp do begin
    rapf[i] := 0 ;
    sigmaf[i] := 0 ;
    rcut[i] := 0 ;
    rhit[i] := 0 ;
    sigcts[i] := 0 ;
    sigfil[i] := 0 ;
  end ;
  for i := 1 to ncp do begin
    sigcrt[i] := 0 ;
    astp[i] := 0 ;
    sigasp[i] := 0 ;
    arep[i] := 0 ;
    enapfl[i] := 0 ;
    snapfl[i] := 0 ;
    ihit[i] := 0 ;
    ipcut[i] := 0 ;
    amin[i] := 0 ;
    apmin[i] := 0 ;
```

```
    apra[i] := 0 ;
    dstr[i] := 0 ;
    sigarp[i] := 0 ;
  end ;
  for i := 1 to ncp + 1 do begin
    i2cut[i] := 0 ;
    icrat[i] := 0 ;
    sgcrat[i] := 0 ;
    smina[i] := 0 ;
    sgmina[i] := 0 ;
    sapr[i] := 0 ;
    sgapr[i] := 0 ;
    sapra[i] := 0 ;
    sgapra[i] := 0 ;
    for j := 1 to ncp do icut[i,j] := 0 ;
  end ;
end ; { initialize }

{==================================================================}
procedure repair ;

  label return, 30, 60 ;

  var   itgttp, j, jwpntp, j1, kk, ktyp, k1, k5, k9, l, nrep : integer ;

        sumr : real ;

begin
  nrep := kz ;
  if nrep > mxptch then nrep := mxptch ;
  if nrep = 0 then goto return ;
  k1 := 0 ;
  k9 := kz ;
  ktyp := irepr mod 10 ;
  if ( ktyp > 0 ) and ( ( save[1,3] < ii3 ) or ( ktyp = 2 ) ) then begin
    if ( save[1,3] > ii3 ) and ( ktyp = 2 ) then goto return ;
    j := 1  ;
    while ( j <= kz ) and ( save[j,3] <= ii3 ) do begin
      if save[j,3] < ii3 then k1 := j ;
      j := j + 1 ;
    end ;
    k9 := j - 1 ;
  end ;
30:
  if k9 > nrep + k1 then k9 := nrep + k1 ;
  k1 := k1 + 1 ;
  if k9 < k1 then begin
    if ktyp = 2 then goto return ;
    k1 := 0 ;
    k9 := kz ;
    goto 30 ;
  end ;
```

```
l := trunc( save[k1,3] + 0.01 ) ;
sumr := kh[l] - k1 + 1 ;
if narea = 0 then sumr := amin[l] ;
if k9 < kh[l] then begin
  sumr := k9 - k1 + 1 ;
  if narea = 0 then begin
    if sumr <= kh[l] - k9 then begin
      sumr := 0 ;
      ovlap( k1, itgt[l,2], xc[l] - crit[l,1], yc[l], trunc( crit[l,1] ),
             trunc( crit[l,2] ), k9 - k1 + 1, sumr ) ;
      goto 60 ;
    end ;
    j := k9 + 1 ;
    sumr := 0 ;
    ovlap ( j, itgt[l,2], xc[l] - crit[l,1] - 2 * crmax, yc[l] - 2 * crmax,
            trunc( crit[l,1] + 4 * crmax ), trunc( crit[l,2] + 4 * crmax ) ,
            kh[l] - k9, sumr ) ;
    sumr := amin[l] - sumr ;
  end ;
end ;
60:
arep[l] := arep[l] + sumr ;
sigarp[l] := sigarp[l] + sqr( sumr ) ;
k5 := kh[l] ;
if k5 > k9 then k5 := k9 ;
k5 := k5 + 1 ;
for j := k5 to m0 do begin
  j1 := k1 + j - k5 ;
  save[j1,1] := save[j,1] ;
  save[j1,2] := save[j,2] ;
  save[j1,3] := save[j,3] ;
  isav[j1] := isav[j] ;
end ;
k5 := k5 - k1 ;
nrep := nrep - k5 ;
mxptch := mxptch - k5 ;
kz := kz - k5 ;
m0 := m0 - k5 ;
for j := 1 to ncp do kh[j] := kh[j] - k5 ;
if ( nrep = 0 ) or ( kz = 0 ) then goto return ;
if ( save[k1,3] <> 1 ) then begin
  if ktyp = 2 then goto return ;
  k1 := 0 ;
  k9 := kz ;
  goto 30 ;
end ;
```

```
{ ----repair hits on approach for lth target -- if appropriate---- }
  j := k1 ;
  while ( save[j,3] = 1 ) and ( j - k1 + 1 <= nrep ) and ( j <= kz ) do begin
    itgttp := itgt[l,2] ;
    jwpntp := isav[j] ;
    if narea = 0 then sumr := sumr + 4 * sqr( crtab[itgttp,jwpntp,1] ) ;
    j := j + 1 ;
  end ;
  k5 := j - k1 ;
  writeln( lst, 'k1 = ', k1, '  kz = ', kz, '  m0 = ', m0, '  j = ',j ) ;
  for kk := 1 to m0 do
    writeln( lst, save[kk,1]:13:2, save[kk,2]:13:2, save[kk,3]:13:2 ) ;
  for j1 := j to m0 do begin
    kk := k1 + j1 - j ;
    save[kk,1] := save[j1,1] ;
    save[kk,2] := save[j1,2] ;
    save[kk,3] := save[j1,3] ;
    isav[kk] := isav[j1] ;
  end ;
  if narea = 1 then sumr := k5 ;
  writeln ( lst ) ;
  writeln( lst, 'Number of craters filled on approach  = ',k5 ) ;
  nrep := nrep - k5 ;
  mxptch := mxptch - k5 ;
  kz := kz - k5 ;
  m0 := m0 - k5 ;
  l := l + 1 ;
  if l <= ncp then for j := 1 to ncp do kh[j] := kh[j] - k5 ;
  if ( nrep = 0 ) or ( kz = 0 ) or ( ktyp = 2 ) then goto return ;
  k1 := 0 ;
  k9 := kz ;
  goto 30 ;
return:
end ; {  procedure repair  }

{=====================================================================}
procedure ncomp ;
{ ----this routine is entered to calculate the minimum sample size
      of monte carlo iterations to give a specific confidence level
      and interval for the probability of cutting a takeoff
      surface. it cannot be entered unless nflag3 is set in main
      program and nsamp specified as greater than 200.           ---- }
  var   setflag : boolean ;
        j, jx, num : integer ;
        (integers i and ix removed; inactive in fortran version)

        pknum, small, small1, ssize : real ;

        pr : array[1..3] of real ;
{ ----calculate and store in a matrix the probability of cut for
      each target element, using this pattern.                  ---- }
```

```
begin
  setflag := true ;
  if ncp >= 1 then begin
    if zalph < 1.645 then zalph := 1.645 ;
    if ( error > 0.05 ) or ( error < 0.0001 ) then error := 0.05 ;
    for j := 1 to ncp do pr[j] := rcut[j] / nsamp ;
{ ------initialize a loop to find that probability of cut closest
        to 0.5. this maximizes required sample size for worst case
        target element and attack.                                  ------ }
    small := abs( pr[1] - 0.5 ) ;
    jx := 1 ;
{ ------loop to find probability of cut closest to 0.5
        and record it as pknum.                           ------ }
    for j := 1 to ncp do begin
      small1 := abs( pr[j] - 0.5 ) ;
      if small1 < small then begin
        jx := j ;
        small := small1 ;
      end ;
    end ;
    pknum := pr[jx] ;
    num := 0 ;
{ ------if pknum is very close to zero or one, the statistics
        collapse monte carlo iterations to a very small number.
        then calculation of additional iterations to run or
        return to the monte carlo loop should not be completed.
        this accomplished by setting nflag1.                 ------ }
    if ( pKnum > 0.0009 ) and ( pknum < 0.9995 ) then begin
{ --------calculate total sample size to assure confidence level
          and error interval.                                ------ }
      ssize := pknum * ( 1 - pknum ) * ( sqr( zalph / error) ) ;
      num := trunc( ssize ) + 1 ;
{ --------test if more iterations required, setting appropriate flags
          whether to return to the monte carlo loop. if so, set lower
          and upper monte carlo loop limits.                     ------- }
      if num > nsamp then begin
        nsampr := nsamp + 1 ;
        nflag2 := 1 ;
        if num < nmax
          then nsamp := num
          else nsamp := nmax ;
        setflag := false ;
      end ;
    end ;
  end ;
  if setflag then nflag1 := 1 ;
end ; { procedure ncomp }
```

```
{==========================================================================}
procedure reslts ;

  var i, ia, ib, iel1, iel2, ipl1, itgtgp, k, kj, kk, km,
        l, lcount, lv, ncpl : integer ;

      conf90, ct, sampl, sampo : real ;

      pr1, pr2, pr3, pr4, pr5, pr6 : array[1..15] of real ;

begin
  lv := 0 ;
  lcount := 0 ;
  sampl := 1 / nsamp2 ;
  sampo := nsamp2 - 1 ;
  for i := 1 to ntgps do begin
    gparea[i] := 0 ;
    gpadm[i] := 0 ;
    gpht[i] := 0 ;
  end ;
  ct := 0 ;
  for l := 1 to nelt do begin
    if countr[l] > ct then begin
      lcount := l ;
      ct := countr[l] ;
    end ;
    itgtgp := itgt[l,3] ;
    gpht[itgtgp] := gpht[itgtgp] + countr[l] ;
    gpadm[itgtgp] := gpadm[itgtgp] + adm[l] ;
    gparea[itgtgp] := gparea[itgtgp] + tgt[l,4] * tgt[l,5] ;
  end ;
  conf90 := sights[lcount] - sampl * sqr( countr[lcount] ) ;
  conf90 := sqrt( conf90 / sampo ) ;
  conf90 := 2.576 * conf90 * sqrt( sampl ) ;
  writeln ( lst ) ;
  writeln( lst, ' nsamp =',nsamp2:5, '      conf interval for 99% level =',
          conf90:7:3, '  for tgt elt = ', lcount ) ;
  conf90 := 1.645 * conf90 / 2.576 ;
  writeln( lst, '                      conf interval for 90% level =',
          conf90:7:3 ) ;
  if ( nflag3 = 1 ) and ( nsamp2 >= 200 )
    then writeln( lst, ' nsamp limited to least of value input or number needed ',
              'to give specified quality to probability of cut.') ;
  ib := 0 ;
  repeat
    ia := ib + 1 ;
    ib := ia + 14 ;
    if ib > nelt then ib := nelt ;
    km := ib - ia + 1 ;
    writeln ( lst ) ;
    write( lst, '  tgt element' ) ;
    for k := ia to ib do write( lst, k:8 ) ;
    writeln( lst ) ;
```

```
for k := 1 to km do begin
  l := k + ia - 1 ;
  pr1[k] := sampl * countr[l] ;
  pr2[k] := sights[l] - sampl * sqr( countr[l] ) ;
  pr2[k] := sqrt( pr2[k] / sampo ) ;
  pr3[k] := sampl * adm[l] ;
  pr4[k] := sigadm[l] - sampl * sqr( adm[l] ) ;
  pr4[k] := sqrt( pr4[k] / sampo ) ;
end ;
write( lst, ' exp no. hits' ) ;
for k := 1 to km do write( lst, pr1[k]:8:3 ) ;
writeln( lst ) ;
write( lst, '       sigma' ) ;
for k := 1 to km do write( lst, pr2[k]:8:3 ) ;
writeln( lst ) ;
if narea = 0 then begin
  write( lst, ' exp area dam' ) ;
  for k := 1 to km do write( lst, pr3[k]:8:0 ) ;
  writeln( lst ) ;
  write( lst, '       sigma' ) ;
  for k := 1 to km do write( lst, pr4[k]:8:0 ) ;
  writeln( lst ) ;
end ;
write( lst, '  tgt gp. no.' ) ;
for k := ia to ib do write( lst, itgt[k,3]:8 ) ;
writeln( lst ) ;
until ib >= nelt ;
writeln ( lst ) ;
writeln( lst, ' target groups' ) ;
ib := 0 ;
repeat
  ia := ib + 1 ;
  ib := ia + 14 ;
  if ib > ntgps then ib := ntgps ;
  km := ib - ia + 1 ;
  write( lst, '  tgt gp. no.' ) ;
  for k := ia to ib do write( lst, k:8 ) ;
  writeln( lst ) ;
  for k := 1 to km do begin
    l := k + ia - 1 ;
    pr1[k] := gphts[l] - sampl * sqr( gpht[l] ) ;
    pr1[k] := sqrt( pr1[k] / sampo ) ;
    gpht[l] := sampl * gpht[l] ;
    pr2[k] := gpadms[l] - sampl * sqr( gpadm[l] ) ;
    pr2[k] := sqrt( pr2[k] / sampo ) ;
    gpadm[l] := sampl * gpadm[l] ;
    gparea[l] := gpadm[l] / gparea[l] ;
  end ;
  write( lst, ' exp no. hits' ) ;
  for k := ia to ib do write( lst, gpht[k]:8:3 ) ;
  writeln( lst ) ;
  write( lst, '       sigma' ) ;
  for k := 1 to km do write( lst, pr1[k]:8:3 ) ;
```

```
  writeln( lst ) ;
  if narea = 0 then begin
    write( lst, ' exp area dam' ) ;
    for k := ia to ib do write( lst, gpadm[k]:8:0 ) ;
    writeln( lst ) ;
    write( lst, '       sigma' ) ;
    for k := 1 to km do write( lst, pr2[k]:8:0 ) ;
    writeln( lst ) ;
    write( lst, ' exp per. dam' ) ;
    for k := ia to ib do write( lst, gparea[k]:8:3 ) ;
    writeln( lst ) ;
  end ;
until ib >= ntgps ;
if ncp > 0 then begin
  writeln( lst ) ;
  writeln( lst, '     for runways and major taxiways' ) ;
  writeln( lst, '       tgt    mcl  mcw   prob  sigma  exp no   sigma',
          '  exp area   sigma     exp  no  sigma   exp appr',
          '  sigma   exp appr   sigma' ) ;
  writeln( lst, '       elt                 cut        craters',
          '             fill               filled           no crat',
          '             fill' ) ;
  writeln( lst ) ;
  for l := 1 to ncp do begin
    pri[1] := sampl * rcut[l] ;
    pri[2] := sqrt( ( pri[1] - sqr( pri[1] ) ) * sampl ) ;
    pri[4] := sigcrt[l] - sampl * sqr( rhit[l] ) ;
    pri[4] := sqrt( pri[4] / sampo ) ;
    pri[3] := sampl * rhit[l] ;
    pri[5] := sampl * astp[l] ;
    pri[6] := sigasp[l] - sampl * sqr( astp[l] ) ;
    pri[6] := sqrt( pri[6] / sampo ) ;
    pri[7] := sampl * arep[l] ;
    pri[8] := sigarp[l] - sampl * sqr( arep[l] ) ;
    pri[8] := sqrt( pri[8] / sampo ) ;
    pri[12] := signaf[l] - sampl * sqr( rapf[l] ) ;
    pri[12] := sqrt( pri[12] / sampo ) ;
    pri[11] := sampl * rapf[l] ;
    pri[10] := snapfl[l] - sampl * sqr( enapfl[l] ) ;
    pri[10] := sqrt( pri[10] / sampo ) ;
    pri[9] := sampl * enapfl[l] ;
    write( lst, l:11, crit[l,1]:7:0, crit[l,2]:5:0, pri[1]:7:3, pri[2]:7:3,
           pri[3]:8:3, pri[4]:8:3 ) ;
    if narea = 1
      then write ( lst, '       N/A    N/A ' )
      else write ( lst, pri[5]:11:0, pri[6]:8:0 ) ;
    if mxptch = 0
      then write ( lst, '      N/A      N/A ' )
      else write ( lst, pri[7]:11:0, pri[8]:8:0 ) ;
    if apprcw < 1
      then write ( lst, '      N/A      N/A        N/A     N/A' )
      else write ( lst, pri[9]:11:3,pri[10]:9:3,pri[11]:11:0,pri[12]:9:0 ) ;
    writeln ( lst ) ;
```

```
end ;
if ncp > 1 then begin
  writeln( lst ) ;
  writeln( lst, '     combined probabilities of cut' ) ;
  writeln( lst, '                                                           ',
          '                          distribution' ) ;
  writeln( lst, '                                                           ',
          '                        minimum  craters' ) ;
  writeln( lst, '       tgt    mcl  mcw   prob  ',
          'sigma  exp no   sigma   exp area   sigma    elt  ',
          'elt   elt   exp appr   sigma   exp appr   sigma' ) ;
  writeln( lst, '      elts                cut       craters                 ',
          'fill             1     2     3     no crat                ',
          'fill' ) ;
  writeln( lst ) ;
  iel1 := 1 ;
  iel2 := 2 ;
  ncp1 := ncp + 1 ;
  kj := 1 ;
  for l := 1 to 3 do dstr[l] := sampl * icut[kj,l] ;
  while kj <= ncp1 do begin
    kk := 4 - kj ;
    pri[1] := sampl * i2cut[kj] ;
    pri[2] := sqrt( sampl * ( pri[1] - sqr( pri[1] ) ) ) ;
    pri[4] := icrat[kj] ;
    pri[3] := sampl * pri[4] ;
    pri[4] := sgcrat[kj] - sampl * sqr( pri[4] ) ;
    pri[4] := sqrt( pri[4] / sampo ) ;
    pri[5] := sampl * smina[kj] ;
    pri[6] := sgmina[kj] - sampl * sqr( smina[kj] ) ;
    pri[6] := sqrt( pri[6] / sampo ) ;
    pri[7] := sampl * sapr[kj] ;
    pri[8] := sgapr[kj] - sampl * sqr( sapr[kj] ) ;
    pri[8] := sqrt( pri[8] / sampo ) ;
    pri[9] := sampl * sapra[kj] ;
    pri[10] := sgapra[kj] - sampl * sqr( sapra[kj] ) ;
    pri[10] := sqrt( pri[10] / sampo ) ;
    if kj < 4
      then begin
        if kj = 2 then iel2 := 3 ;
        if kj = 3 then iel1 := 2 ;
        write( lst, '        ',iel1,'&', iel2, crit[iel1,1]:7:0,
              crit[iel2,2]:5:0, pri[1]:7:3, pri[2]:7:3, pri[3]:8:3,
              pri[4]:8:3 ) ;
        if ncp <> 3 then kj := ncp1 + 1 ;  { end kj loop }
       end
      else write( lst, '      1&2&3', crit[1,1]:7:0, crit[1,2]:5:0,
              pri[1]:7:3,pri[2]:7:3,pri[3]:8:3,pri[4]:8:3 ) ;
    if narea = 1
      then write( lst, '      N/A      N/A   ' )
      else write( lst, pri[5]:11:0, pri[6]:8:0, '  ' ) ;
```

```
          for l := 1 to 3 do if l = kk
            then write( lst, ' N/A ' )
            else write( lst, dstr[l]:6:3 ) ;
          if apprcw < 1
            then writeln( lst, '    N/A      N/A       N/A     N/A' )
            else writeln( lst, pr1[7]:10:3, pr1[8]:8:3, pr1[9]:11:0,
                          pr1[10]:8:0 ) ;
          kj := kj + 1 ;
        end ; { kj loop }
      end ;
    end ;
    writeln ( lst ) ;
    lv := 0 ;
    for l := 1 to nelt do
      if ( itgt[l,1] = 1 ) and ( crit[l,1] < 1 ) then begin
        lv := lv + 1 ;
        ipl[lv] := l ;
      end ;
    if lv > 0 then begin
      writeln( lst, '    for minor taxiways' ) ;
      ib := 0 ;
      repeat
        ia := ib + 1 ;
        ib := ia + 14 ;
        if ib > lv then ib := lv ;
        km := ib - ia + 1 ;
        writeln ( lst ) ;
        write( lst, '             target element' ) ;
        for k := ia to ib do write( lst, ipl[k]:7 ) ;
        writeln( lst ) ;
{ -------non-ansi standard subscripts may require adjustment. ------- }
        write( lst, '               target width' ) ;
        for k := ia to ib do write( lst, tgt[ipl[k],5]:7:0 ) ;
        writeln( lst ) ;
        write( lst, '        minimum clear width' ) ;
        for k := ia to ib do write( lst, crit[ipl[k],2]:7:0 ) ;
        writeln( lst ) ;
        for k := 1 to km do begin
          l := k + ia - 1 ;
          ipll := ipl[l] ;
          pr1[k] := sampl * rcut[ipll] ;
          pr2[k] := sigcts[l] - sampl * sqr( rcut[ipll] ) ;
          pr2[k] := sqrt( pr2[k] / sampo ) ;
          pr3[k] := sampl * rhit[ipll] ;
          pr4[k] := sigfil[l] - sampl * sqr( rhit[ipll] ) ;
          pr4[k] := sqrt( pr4[k] / sampo ) ;
          pr6[k] := sampl * rapf[ipll] ;
          pr5[k] := signaf[ipll] - sampl * sqr( rapf[ipll] ) ;
          pr5[k] := sqrt( pr5[k] / sampo ) ;
        end ;
        write( lst, '    expected number of cuts' ) ;
        for k := 1 to km do write( lst, pr1[k]:7:3 ) ;
        writeln( lst ) ;
```

```
      write( lst, '                        sigma' ) ;
      for k := 1 to km do write( lst, pr2[k]:7:3 ) ;
      writeln( lst ) ;
      write( lst, '   expected craters to fill' ) ;
      for k := 1 to km do write( lst, pr3[k]:7:3 ) ;
      writeln( lst ) ;
      write( lst, '                        sigma' ) ;
      for k := 1 to km do write( lst, pr4[k]:7:3 ) ;
      writeln( lst ) ;
      if narea = 0 then begin
        write( lst, '      expected area to fill' ) ;
        for k := 1 to km do write( lst, pr6[k]:7:0 ) ;
        writeln( lst ) ;
        write( lst, '                        sigma' ) ;
        for k := 1 to km do write( lst, pr5[k]:7:0 ) ;
        writeln( lst ) ;
      end ;
    until ib >= lv ;
  end ;
end ; {  procedure reslts  }

{=================================================================}
procedure query( var ichc: integer ) ;
{       this procedure returns the response to a yes-no question
        ichc is set to 0 for yes, and 1 for no or ( return ).      }

  var ans : char ;
begin
  write( '  (y/n)  ==>  ' ) ;
  repeat readln( ans ) until ( upcase( ans ) in [ 'Y','N' ] ) or eoln ;
  writeln ;
  if eoln or ( upcase( ans ) = 'N' )
    then ichc := 1
    else ichc := 0 ;
end ;  { procedure query }

{=================================================================}
procedure pause ;
{    delays crt output until a key is pressed     }
begin
  writeln ;
  write( '          p r e s s   a n y   k e y   t o   c o n t i n u e  ' ) ;
  repeat until keypressed ;
  clrscr ;
end ;  {  procedure pause  }
```

```
{=================================================================}
procedure indat ;
{      this procedure prepares the output file and reads the input file
       for AAPMOD.  }
  type str14 = string[14] ;

  var goodfile : boolean ;
      i, j, jr, yn : integer ;
      fname1, fname2 : str14 ;
      diskfile : text ;

  procedure processfile( var fname : str14; which : str14 ) ;
  { initial processing of file name entered by user }
    var dot, goodname : boolean ;
      p, size, dotpos : integer ;
  begin
    clrscr ;
    writeln ;
    writeln( 'For the following entries, allowable filenames may contain ',
             'only capital letters' ) ;
    writeln( 'and numbers.  If you enter lower case letters, they will be ',
           'treated as if they' ) ;
    writeln( 'are upper case.  The format must be as follows:' ) ;
    writeln ;
    writeln( '          D:XXXXXXXX.XXX' ) ;
    writeln ;
    writeln( '   where   D: is an optional drive specifier,' ) ;
    writeln( '           X is a digit ( 0 - 9 ) or a letter ( A - Z ),' ) ;
    writeln( '           XXXXXXXX is a filename 1 to 8 characters long, and' );
    writeln( '           .XXX is an optional 1 to 3 character file type',
           ' ( not .BAK ).' ) ;
    writeln ;
    writeln( 'Examples of valid filenames:  MSN   MSN.1   AAPMOD.DTA   ',
             'B:1JAN84.AAP' ) ;
    writeln ;
    writeln ;
    writeln( 'If an existing file you need to use contains  characters  not ',
             'compatible with' ) ;
    writeln( 'the above format,  you will need to ABORT this program by ',
             'pressing control-C,' ) ;
    writeln( 'and then rename the file to make it compatible.' ) ;
    pause ;
    repeat
      goodname := true ;
      dot := false ;
      repeat
        writeln ;
        write( 'Enter name for the ', which, ' File  ==>  ' ) ;
        readln( fname ) ;
        writeln ;
        for p := 1 to 14 do fname[p] := upcase( fname[p] ) ;
```

```
  repeat                                   { delete blanks from }
    p := pos( ' ', fname ) ;               { filename           }
    if p <> 0 then delete( fname, p, 1 ) ;
  until p = 0 ;
until fname <> '' ;  { trap for carriage return/blanks only }
dotpos := pos( '.', fname ) ; { filetype can only be 3 characters long }
if dotpos <> 0 then delete( fname, dotpos + 4, 14 ) ;
size := length( fname ) ;
p := pos( ':', fname ) ;
if ( p = 2 )
  then begin

      if ( not ( dotpos in [ 0, 4..11 ] ) )
                    or
         ( not ( fname[1] in [ 'A'..'Z' ] ) )
                    or
                ( size < 3 )
                    or
         ( ( dotpos = 0 ) and ( size > 10 ) )
                   then
              goodname := false ;

      if goodname then for p := 3 to size do begin
        if not ( fname[p] in [ '0'..'9', 'A'..'Z', '.' ] )
          then goodname := false ;
        if dot and ( fname[p] = '.' ) then goodname := false ;
        if fname[p] = '.' then dot := true ;
      end ;
    end
  else begin

      if ( not ( dotpos in [ 0, 2..9 ] ) )
                    or
                ( size < 0 )
                    or
         ( ( dotpos = 0 ) and ( size > 8 ) )
                   then
              goodname := false ;

      if goodname then for p := 1 to length( fname ) do begin
        if not ( fname[p] in [ '0'..'9', 'A'..'Z', '.' ] )
          then goodname := false ;
        if dot and ( fname[p] = '.' ) then goodname := false ;
        if fname[p] = '.' then dot := true ;
      end ;
    end ;
if pos( '.BAK', fname ) <> 0 then begin
  goodname := false ;
  writeln( 'You may not enter a filename with the filetype ".BAK" .' ) ;
  writeln ;
  writeln( 'To use the backup .BAK file, rename it before executing ',
           'this program.' ) ;
  writeln ;
```

```
    end ;
    writeln ;
    if goodname
      then begin
          write( 'Confirm the correct filename is < ', fname, ' > ?' ) ;
          query( p ) ;
          if p = 1 then goodname := false ;
        end
      else begin
          writeln( 'Invalid filename < ',fname, ' >' ) ;
          writeln ;
          writeln( 'Please try again, or press control-C to ABORT this ',
                   'program.' ) ;
        end ;
  until goodname ;
end ;  { procedure processfile }

{==============================================================================}
function filexists( fname : str14 ) : boolean ;
{ Checks to see if file fname exists on disk.  Returns true if it does. }
  var exists : boolean ;
begin
  assign( diskfile, fname ) ;
  {$I-} reset( diskfile ) ; {$I+  this checks if the file exists}
  exists := ioresult = 0 ;
  if not exists then begin
    writeln( 'File < ', fname, ' > does not exist.  Be sure to specify the ',
             'correct drive ' ) ;
    writeln( 'and the correct file name.' ) ;
    writeln ;
    writeln( '                 To ABORT this program, press control-C.' ) ;
    pause ;
  end ;
  filexists := exists ;
end ;  { function filexists }

begin  { body of function indat }
  clrscr ;
  writeln ;
  writeln ;
  writeln( '                              W E L C O M E' ) ;
  writeln ;
  writeln( '                               t o   t h e' ) ;
  writeln ;
  writeln( '  A I R F I E L D   A T T A C K   A S S E S S M E N T  ',
         'P R O G R A M' ) ;
  writeln( '  = = = = = = = =   = = = = = =   = = = = = = = = = =  ',
         '= = = = = = =' ) ;
  writeln ;
  writeln ;
  writeln( '      This program was rewritten in TURBO PASCAL ( Version ',
         '2.1 )' ) ;
  writeln ;
```

```
writeln( '                              by' ) ;
writeln ;
writeln( '   Major David A. Roodhouse, USAF and Captain Thomas K. Green, ',
        'USAF' ) ;
writeln ;
writeln( '                AIR FORCE INSTITUTE OF TECHNOLOGY' ) ;
writeln ;
writeln( '               WRIGHT-PATTERSON AIR FORCE BASE, OHIO' ) ;
writeln ;
writeln( '                       February 1985' ) ;
writeln ;
pause ;
writeln ;
writeln ;
writeln( 'First you will need to enter the name of the file containing the ',
        'input data' ) ;
writeln ;
writeln( 'for this program.  This could be a file made by running the ',
        'program called' ) ;
writeln ;
writeln( 'AAPMSN, or you may be using some other existing input file.   ',
        'Then you need' ) ;
writeln ;
writeln( 'to select the name for the new output file to be generated by ',
        'AAPMOD.' ) ;
writeln ;
writeln ;
pause ;
repeat processfile( fname1, 'AAPMOD Input' ) until filexists( fname1 ) ;
writeln( 'Reading AAPMOD input file from disk file:  ( ', fname1, ' )' ) ;
assign( diskfile, fname1 ) ;
reset( diskfile ) ;
readln( diskfile ) ;    ( Ignore seed in the input file )
read( diskfile, nsamp, nsampt, nflag3, error, zalph, nelt, ntgps, apprcw,
      narea ) ;
for i := 1 to nelt do begin
  for j := 1 to 5 do read( diskfile, tgt[i,j] ) ;
  for j := 1 to 3 do read( diskfile, itgt[i,j] ) ;
  if itgt[i,1] = 1
    then read( diskfile, crit[i,1], crit[i,2] )
    else begin
         crit[i,1] := 0 ;
         crit[i,2] := 0 ;
       end ;
  tgt[i,3] := tgt[i,3] * pi / 180 ;  ( convert degrees to radians )
end ;
read( diskfile, ncp, lv, npatt ) ;
for i := 1 to npatt do begin
  for j := 1 to 4 do read( diskfile, ipat[i,j] ) ;
  for j := 1 to 10 do read( diskfile, patt[i,j] ) ;
  for j := 1 to ipat[i,1] do begin
    jr := 2 * j + 9 ;
    read( diskfile, patt[i,jr], patt[i, jr + 1] ) ;
```

```
    end ;
  end ;
  read( diskfile, m, n ) ;
  for i := 1 to m do begin
    for j := 1 to n do read( diskfile, crtab[i,j,1] ) ;
    for j := 1 to n do read( diskfile, crtab[i,j,2] ) ;
  end ;
  read( diskfile, mxptch, irepr, npass ) ;
  for i := 1 to npass do begin
    for j := 1 to 5 do read( diskfile, pass[i,j] ) ;
    read( diskfile, ipass[i,1], ipass[i,2], npx[i] ) ;
    pass[i,3] := pass[i,3] * pi / 180 ;  { convert degrees to radians }
    pass[i,6] := pass[i,5] ;
  end ;
  close( diskfile ) ;
  repeat
    processfile( fname2, 'New Output' ) ;
    assign( lst, fname2 ) ;
    {$I-} reset( lst ) ; {$I+  see if file already exists }
    goodfile := ioresult <> 0 ;  { want the file not to exist }
    if not goodfile then begin
      write( 'File < ', fname2, ' > already exists.  Erase?' ) ;
      query( yn ) ;
      if yn = 0 then begin
        write( 'Are you SURE you want to erase file < ',fname2, ' > ?' ) ;
        query( yn ) ;
        goodfile := yn = 0 ;
      end ;
    end ;
  until goodfile ;
  rewrite( lst ) ;
  writeln( lst, fname1, ' was the input file for the following run.' ) ;
  writeln( lst ) ;
  clrscr ;
  writeln ;
  writeln ;
  writeln( 'The output file is called "', fname2, '"' ) ;
  writeln ;
  writeln ;
  writeln( 'N U M B E R   C R U N C H I N G   I N   P R O G R E S S   !' ) ;
  writeln ;
  end ;  { procedure iadat }
```

```
begin { =============== M A I N   P R O G R A M =============== }

{ --- input/initialize --- }

indat ;  { read input file and set up output file }
initialize ;

{  test to see if limiting monte carlo loops is both desired (nflag3=1)
   and appropriate (nsamp > 200).  if so, set flags and set initial
   monte carlo loop limit.}

if ( (nflag3 = 1) and (nsamp >= 200) ) then begin
  nflag1 := 0 ;
  nflag2 := 0 ;
  nmax := nsamp ;
  nsamp := 200 ;
end ;
{    ---monte carlo loop  --  820 on (it) ---}
85:
for it := nsampr to nsamp do begin
{  ---initialize variables which get reset each monte carlo rep --- }
  nsamp2 := it ;
  for l := 1 to nelt do decar[l] := tgt[l,4] * tgt[l,5] ;
  for l := 1 to 3 do begin
    ipcut[l] := 0 ;
    ihit[l] := 0 ;
    amin[l] := 0 ;
    aprmin[l] := 0 ;
    apra[l] := 0 ;
  end;
  m := 0 ;
  m0 := 0 ;
  Kz := 0 ;
{  ---set number of hits per target equal to zero--- }
  for l := 1 to nelt do lnhits[l] := 0 ;
{  ---compute impact points of weapons--- }
  for i := 1 to npass do begin
{  -----see if a/c survived. if yes, change next pass ps to reattack ps
                              if not, change next pass ps to 0.0,
                                      and log no hits for this pass -----}
    nxtp := ipass[i,2] ;
    if random >= pass[i,4]
      then begin
          pass[nxtp,4] := 0 ;
          goto 370 ;
        end
      else pass[nxtp,4] := pass[i,5] ;
    nptrn := ipass[i,1] ;
    nwep := ipat[nptrn,1] ;
    nbom := ipat[nptrn,2] ;
    rmaj := patt[nptrn,5] ;
    rmin := patt[nptrn,6] ;
    vmaj := patt[nptrn,7] ;
```

```
    vmin := patt[nptrn,8] ;
    kode := ipat[nptrn,4] ;
{  ---locate stick pattern center--- }
    passxt := pass[i,1] ;
    passyt := pass[i,2] ;
    if npx[i] <= ncp then begin
      nttt := npx[i] ;
      trisub(dap) ;
      passxt := passxt + dap * cos( tgt[nttt,3] ) ;
      passyt := passyt + dap * sin( tgt[nttt,3] ) ;
    end ;
    sinp := sin( pass[i,3] ) ;
    cosp := cos( pass[i,3] ) ;
    if kode = 3
      then begin  { - - - - -guided munitions... }
          crazyn := random ;
          if crazyn <= patt[nptrn,7]
            then noran (r, patt[nptrn,1], d, patt[nptrn,2] )
            else if crazyn <= patt[nptrn,8]
              then noran (r, patt[nptrn,3], d, patt[nptrn,4] )
              else noran (r, patt[nptrn,5], d, patt[nptrn,6] ) ;
          x := passxt + r * cosp + d * sinp ;
          y := passyt + r * sinp - d * cosp ;
        end  {   guided munitions   }
      else begin  { - - - - -dumb bombs...  }
          noran (r,patt[nptrn,1],d,patt[nptrn,2]) ;
          xctr := passxt + r * cosp + d * sinp ;
          yctr := passyt + r * sinp - d * cosp ;
        end ;
{ ------locate weapon impact or center of dispenser pattern------ }
    for k := 1 to nwep do begin
      if random  > patt[nptrn,9] then goto 360 ;
      if kode < 3 then begin
        noran( r, patt[nptrn,3], d, patt[nptrn,4] ) ;
        k2 := 2 * k + 9 ;
        xiwod := xctr + ( patt[nptrn,k2] + r ) * cosp
                      + ( patt[nptrn,k2+1] + d ) * sinp ;
        yiwod := yctr + ( patt[nptrn,k2] + r ) * sinp
                      - ( patt[nptrn,k2+1] + d ) * cosp ;
      end ;
{ -------locate impacts  (nbom = 1 or nmbr bomblets/cbu shell)------- }
      for m1 := 1 to nbom do begin
        if kode < 3 then begin
          x := xiwod ;
          y := yiwod ;
          if nbom > 1 then begin
            if random > patt[nptrn,10] then goto 350 ;
280:        x1 := 2 * rmaj * random - rmaj ;
            y1 := 2 * rmin * random - rmin ;
            if kode = 2 then begin
              x1y1o1 := ( sqr(x1) / sqr(rmaj) ) + ( sqr(y1) / sqr(rmin) ) ;
              if x1y1o1 > 1 then goto 280 ;
```

```
             if ( vmaj > 0 ) and ( vmin > 0 ) then begin
                 xlyli1 := ( sqr(xl)/sqr(vmaj) ) + ( sqr(yl)/sqr(vmin) ) ;
                 if xlyli1 < 1 then goto 288 ;
               end ;
             end ;
             x := x + xl * cosp + yl * sinp ;
             y := y + xl * sinp - yl * cosp ;
           end ;
         end ;
{ --------check for any hit or near-miss-------- }
         for l := 1 to nelt do begin
           sint := sin(tgt[l,3]) ;
           cost := cos(tgt[l,3]) ;
           xp := x - tgt[l,1] ;
           yp := y - tgt[l,2] ;
           tl := xp * cost + yp * sint ;
           xp := yp * cost - xp * sint ;
           itgttp := itgt[l,2] ;
           jwpntp := ipat[nptrn,3] ;
           if ( l > ncp ) and ( l <= lv + ncp )
             then begin
                 if ( abs(tl) - crtab[itgttp,jwpntp,2] >= 0.5 * tgt[l,4] )
                                                          or
                    ( abs(xp) - crtab[itgttp,jwpntp,2] >= 0.5 * tgt[l,5] )
                   then goto 340 ;
               end
             else begin
                 if ( abs(tl) - crtab[itgttp,jwpntp,1] >= 0.5 * tgt[l,4] )
                                                          or
                    ( abs(xp) - crtab[itgttp,jwpntp,1] >= 0.5 * tgt[l,5] )
                   then goto 340 ;
               end ;
           m := m + 1 ;
           if m <= 800 then begin
             save[m,1] := tl + 0.5 * tgt[l,4] ;
             save[m,2] := xp + 0.5 * tgt[l,5] ;
             save[m,3] := l ;
             isav[m] := ipat[nptrn,3] ;
             countr[l] := countr[l] + 1 ;
             lnhits[l] := lnhits[l] + 1 ;
           end ;
340:     end ; { nelt-loop }
         if m > 800 then begin
           writeln( lst,'More than 800 hits were found in pass',i:4,'.' ) ;
           writeln( lst, ' Excess were ignored.' ) ;
           m := 800 ;
         end ;
350:
       end ; { ml loop }
360:
     end ; { k loop }
370:
   end ; { i loop }
```

```
  if m = 0 then begin
    writeln( lst,' No hits during attack, monte carlo iteration: ',it:4 ) ;
    goto 810 ;
  end;
{ -----mission is complete----- }
  sort(m, 1, 3, 1, 2) ;
  for j := 1 to ntgps do begin
    gpadac[j] := 0 ;
    gphtac[j] := 0 ;
  end ;
  for i := 1 to nelt do begin
    itgtgp := itgt[i,3] ;
    gphtac[itgtgp] := gphtac[itgtgp] + lnhits[i] ;
    sights[i] := sights[i] + sqr(lnhits[i]) ;
  end ;
  for j := 1 to ntgps do gphts[j] := gphts[j] + sqr(gphtac[j]) ;
{ -----initial conditions for bda }
  l := 1 ;
  k0 := 1 ;
  cuts := 0 ;
  fill := 0 ;
  arfill := 0 ;
  arfils := 0 ;
  sumrun := 0 ;
{ -----loop on each impact, and group on target element----- }
  for k := 1 to m do begin
    if ( save[k,3] > l ) or ( k = m ) then begin
430:
      n := k - k0 ;
      if save[k,3] = l then n := n + 1 ;
{ ---------damage assessment--------- }
      if itgt[l,1] = 0
        then begin
{ -----------tgt l is a building or non-pavement----------- }
          if ( n > 0 ) and ( decar[l] > 0 ) then bldg ;
          diffar := tgt[l,4] * tgt[l,5] - decar[l] ;
          adm[l] := adm[l] + diffar ;
          sigadm[l] := sigadm[l] + sqr(diffar) ;
          itgtgp := itgt[l,3] ;
          gpadac[itgtgp] := gpadac[itgtgp] + diffar ;
        end
      else begin
{ ---------tgt l is a pavement--------- }
          if n < 1 then begin
            if l <= ncp then begin
              xc[l] := 0.5 * ( tgt[l,4] + crit[l,1] ) ;
              yc[l] := 0.5 * ( tgt[l,5] - crit[l,2] ) ;
            end ;
            goto 730 ;
          end ;
          sort( n, k0, 1, 2, 3 ) ;
```

```
          if narea = 0
          then ovlap( k0, itgt[l,2], 0, 0, trunc( tgt[l,4] ),
                      trunc( tgt[l,5] ), n, sumrun ) ;
{ ----------taxiways (minor pavements)
          find wandering path only for taxi-only tgts  (crit[l,1]=0 ) ----- }
          if crit[l,1] < 1
            then begin
               mincw(n, k0, 1, 2, itgt[l,2], crit[l,2], tgt[l,5] ) ;
               arfils := arfill ;
               fill := nfill ;
               rhit[l] := rhit[l] + fill ;
               ntxwy := l - ncp ;
               sigfil[ntxwy] := sigfil[ntxwy] + sqr(fill) ;
               rcut[l] := rcut[l] + cuts ;
               sigcts[ntxwy] := sigcts[ntxwy] + sqr(cuts) ;
             end
            else begin
{ ----------runways (major pavements)
    search for a clear strip (length=crit[l,1] .x. width=crit[l,2]) -------- }
               m0 := k - 1 ;
               if ( k = m ) and ( save[m,3] = 1 ) then m0 := m0 + 1 ;
               ipcut[l] := 0 ;
               ihit[l] := 0 ;
               amin[l] := 0 ;
               aprmin[l] := 0 ;
               apra[l] := 0 ;
               for kk := 1 to 4 do
                 for kk2 := 1 to 2 do
                   numapr[kk,kk2] := 0 ;
               clstrp ;
               if nmin > 0 then begin
                 rcut[l] := rcut[l] + 1 ;
                 ipcut[l] := 1 ;
               end ;
               rhit[l] := rhit[l] + nmin ;
               ihit[l] := nmin ;
               sumstp := 0 ;
               kml := k - 1 ;
               if ( k = m ) and ( save[k,3] = 1 ) then kml := k ;
               ka := 1 ;
               mflag := 0 ;
               xs1 := xc[l] ;
               ys1 := yc[l] ;
               xs2 := xc[l] - crit[l,1] ;
               ys2 := yc[l] + crit[l,2] ;
               kh[l] := kz ;
               kp1 := k0 ;
```

```
560:        if ( kp1 <= m ) and ( km1 <= m ) then begin
              itgttp := itgt[l,2] ;
              for kw := kp1 to km1 do begin
                jwpntp := isav[kw] ;
                if save[kw,1] + crtab[itgttp,jwpntp,ka] <= xs2
                  then goto 580 ;
                if save[kw,1] - crmax >= xs1 then goto 590 ;
                if ( save[kw,1] - crtab[itgttp,jwpntp,ka] >= xs1 )
                                              or
                   ( save[kw,2] + crtab[itgttp,jwpntp,ka] <= ys1 )
                                              or
                   ( save[kw,2] - crtab[itgttp,jwpntp,ka] >= ys2 )
                  then goto 580 ;
                kz := kz + 1 ;
                if kw <> kz then begin
                  s1 := save[kw,1] ;
                  s2 := save[kw,2] ;
                  s3 := save[kw,3] ;
                  itt := isav[kw] ;
                  kzp := kz + 1 ;
                  for k8 := kzp to kw do begin
                    kk := kw - k8 + kzp ;
                    save[kk,1] := save[kk-1,1] ;
                    save[kk,2] := save[kk-1,2] ;
                    save[kk,3] := save[kk-1,3] ;
                    isav[kk] := isav[kk-1] ;
                  end ;  {  k8 loop  }
                  save[kz,1] := s1 ;
                  save[kz,2] := s2 ;
                  save[kz,3] := s3 ;
                  isav[kz] := itt ;
                end ;
580:          end ;  {  kw loop  }
            end ;
590:        if mflag = 0 then begin
              kzt := 0 ;
              if kz <> kh[l] then begin
                kzt := kz - kh[l] ;
                kk := kh[l] + 1 ;
                kh[l] := kz ;
                if narea <= 0
                  then ovlap( kk, itgt[l,2], xc[l] - crit[l,1], yc[l],
                              trunc( crit[l,1] ), trunc( crit[l,2] ),
                              kzt, sumstp ) ;
                astp[l] := astp[l] + sumstp ;
                sigasp[l] := sigasp[l] + sqr(sumstp) ;
                amin[l] := sumstp ;
              end ;
              mflag := 1 ;
              ka := 2 ;
              kp1 := kp1 + kzt ;
              kz := kp1 - 1 ;
              kz1 := kz ;
```

```
          xs1 := xc[1] - crit[1,1] ;
          if xs1 >= crit[1,2]
            then begin
                xs2 := crit[1,2] ;
                goto 560 ;
              end
            else goto 640 ;
        end ; ( if at label 590  )
        kzt := 0 ;
        nfill := 0 ;
        if kz <> kz1 then begin
          kzt := kz - kz1 ;
          kk := kz1 + 1 ;
          if mflag >= 3
            then sort(kzt, kk, 2, 1, 3) ;
          for ii := 1 to kzt do begin
            save[kk + ii - 1,2] := save[kk + ii - 1,2] - ys1 ;
            save[kk + ii - 1,1] := save[kk + ii - 1,1] - xs2 ;
          end ;
          if mflag <= 2
            then mincw(kzt, kk, 1, 2, itgt[1,2], apprcw, crit[1,2] )
            else mincw(kzt, kk, 2, 1, itgt[1,2], apprcw, crit[1,2] );
          for ii := 1 to kzt do begin
            save[kk + ii - 1,2] := save[kk + ii - 1,2] + ys1 ;
            save[kk + ii - 1,1] := save[kk + ii - 1,1] + xs2 ;
          end ;
          arfils := arfils + arfill ;
        end ;
        numapr[mflag,2] := kzt ;
        kz := kz1 + nfill ;
        kz1 := kz ;
        numapr[mflag,1] := nfill ;
        fill := fill + nfill ;
        if kz = km1 then goto 670 ;
        case mflag of
            1 : goto 640 ;
            2 : goto 650 ;
            3 : goto 660 ;
            4 : goto 670 ;
        end ; ( case )
640:    mflag := 2 ;
        nfill := 0 ;
        kp1 := kp1 + kzt ;
        xs1 := tgt[1,4] - crit[1,2] ;
        if xc[1] + crit[1,2] <= tgt[1,4] then begin
          xs2 := xc[1] ;
          goto 560 ;
        end ;
650:    mflag := 3 ;
        kp1 := kp1 - numapr[1,2] + numapr[1,1] + nfill ;
        (call) sort(k - kp1, kp1, 1, 2, 3 ) ;
        xs1 := crit[1,2] ;
        ys1 := 0 ;
```

```
                    xs2 := 0 ;
                    ys2 := yc[l] + crit[l,2] ;
                    goto 560 ;
660:                mflag := 4 ;
                    kpl := kpl + kzt ;
                    xs1 := tgt[l,4] ;
                    xs2 := tgt[l,4] - crit[l,2] ;
                    goto 560 ;
670:                kz := kh[l] ;
                    if ( irepr >= 10 ) and ( fill > 0 ) then begin
                      writeln( lst, ' target ',l:3,' kh(l) = ',kh[l]:4,
                              ' k0 = ',k0:4,' fill = ',fill:7:0 ) ;
                      for kk := 1 to m do
                      writeln( lst, ' ',save[kk,1]:10:2,save[kk,2]:10:2,
                               save[kk,3]:10:2 ) ;
                      kz := kh[l] + trunc(fill + 0.01) ;
                      if l > 1 then begin
                        k8 := trunc(fill + 0.01) ;
                        for kz1 := 1 to k8 do begin
                          kk := k0 + kz1 - 1 ;
                          s1 := save[kk,1] ;
                          s2 := save[kk,2] ;
                          s3 := save[kk,3] ;
                          is := isav[kk] ;
                          kzp := kh[l] + kz1 + 1 ;
                          for kw := kzp to kk do begin
                            kw1 := kk - kw + kzp ;
                            save[kw1,1] := save[kw1 - 1, 1] ;
                            save[kw1,2] := save[kw1 - 1, 2] ;
                            save[kw1,3] := save[kw1 - 1, 3] ;
                            isav[kw1] := isav[kw1 - 1] ;
                          end ;
                          kzp := kzp - 1 ;
                          save[kzp,1] := s1 ;
                          save[kzp,2] := s2 ;
                          save[kzp,3] := s3 ;
                          isav[kzp] := is ;
                        end ;
                      end ;
                    end ; {  if near label 670  }
                    sigcrt[l] := sigcrt[l] + sqr(nmin) ;
                    emapfl[l] := emapfl[l] + fill ;
                    smapfl[l] := smapfl[l] + sqr(fill) ;
                    aprmin[l] := fill ;
                    goto 720 ;
                  end ;  {  if crit[l,1] < 1 else begin  }
720:            adm[l] := adm[l] + sumrun ;
                itgtgp := itgt[l,3] ;
                gpadac[itgtgp] := gpadac[itgtgp] + sumrun ;
                sigadm[l] := sigadm[l] + sqr(sumrun) ;
                rapf[l] := rapf[l] + arfils ;
                signaf[l] := signaf[l] + sqr(arfils) ;
                if crit[l,1] > 0 then apra[l] := arfils ;
```

```
        end ; { else: target 1 is pavement  }
730:
      l := l + 1 ;
      k0 := k ;
      fill := 0 ;
      arfill := 0 ;
      arfils := 0 ;
      cuts := 0 ;
      sumrun := 0 ;
      if ( save[k,3] > 1 ) or ( ( k = m ) and ( save[k,3] = 1 ) ) or
         ( ( l <= nelt ) and ( k = m ) ) then goto 430 ;
    end ; {  if, at start of k-loop  }
  end ; { k-loop  }
  for j := 1 to ntgps do gpadms[j] := gpadms[j] + sqr(gpadac[j]) ;
  ii3 := 1 ;
{ -----compute combined probabilities for runway, taxiway,and sod----- }
  if ncp > 1 then begin
    i3 := 0 ;
    kj := 1 ;
    ifin := 0 ;
    for jj := 1 to 2 do
      for jk := 1 to ncp do begin
{ ---------only interested in 1&2 (kj=1), 1&3 (kj=2), 2&3 (kj=3)-------- }
        if jj >= jk then goto 790 ;
        if ipcut[ii3] = 0 then goto 760 ;
        if ipcut[jj] <> 1 then ii3 := jj ;
        if ipcut[jk] <> 1 then ii3 := jk ;
760:    if ( ipcut[jj] <> 1 ) or ( ipcut[jk] <> 1 ) then goto 780 ;
{ ---------both surfaces are cut--------- }
        i3 := i3 + 1 ;
{ ----ii indicates which surface has the minimum number of craters to
      repair for combinations of 2 surfaces and ii3 for all 3 surfaces---- }
        ii := jj ;
        if ihit[jj] > ihit[jk] then ii := jk ;
        if ihit[ii3] > ihit[jk] then ii3 := jk ;
{ ---------distribution of minimum number of craters----------- }
770:    icut[kj,ii] := icut[kj,ii] + 1 ;
        i2cut[kj] := i2cut[kj] + 1 ;
        sgcrat[kj] := sgcrat[kj] + sqr(ihit[ii]) ;
{ ---------minimum number of craters--------- }
        icrat[kj] := icrat[kj] + ihit[ii] ;
{ ---------area of craters---------- }
        smina[kj] := smina[kj] + amin[ii] ;
        sgmina[kj] := sgmina[kj] + sqr(amin[ii]) ;
{ ---------minimum number of craters on approach to operational strip--------- }
        sapr[kj] := sapr[kj] + aprmin[ii] ;
        sgapr[kj] := sgapr[kj] + sqr(aprmin[ii]) ;
{ ---------area of craters on approach---------- }
        sapra[kj] := sapra[kj] + apra[ii] ;
        sgapra[kj] := sgapra[kj] + sqr(apra[ii]) ;
        if ifin = 1 then goto 800 ;
780:    kj := kj + 1 ;
        if ( jj <> 2 ) or ( jk <> 3 ) then goto 790 ;
```

```
{ ----all combinations of 2 surfaces have been looked at. if all 3
      surfaces have been cut (i3=3) compute statistics for all 3 & exit
      loop (ifin=1)---- }
        if i3 <> 3 then goto 888 ;
        kj := 4 ;
        ii := ii3 ;
        ifin := 1 ;
        goto 770 ;
790:  end ; {  jk-loop  }
  end ; {  if ncp > 1  }
888:
  repair ;
  m := m0 ;
  m0 := 0 ;
  kz := 0 ;
810:
  if it mod nsampt = 0 then reslts ;
end ; {  it-loop  }
{ ----test to see if limiting monte carlo loop was desired
      and appropriate. if not, avoid subroutine "ncomp"---- }
if ( nflag3 = 1 ) and ( nsamp >= 200 ) then begin
{ ----tests on flags set inside subroutine "ncomp" to direct
      either return to monte carlo loop or pass on, based on
      estimate of iterations required---- }
  if nflag2 = 0 then ncomp ;
  if nflag1 = 0 then begin
    nflag1 := 1 ;
    goto 85 ;
  end ;
end ;
{ ----calculate and print statistics---- }
if it mod nsampt <> 0 then reslts ;
close( lst ) ;   { needed due to assigning lst as a disk file }
end.
```

```
{==========================================================================}
{===================  file AAPTGT.PAS    18 Feb 85  ========================}
{==========================================================================}

{$IAAPTGT1.PAS  compiler directive to include file AAPTGT1.PAS}
```

```
{===========================================================================}
{==============   FILE AAPTGT1.PAS    10 Feb 85   ==========================}
{===========================================================================}
program aaptgt ;

  type str14 = string[14] ;            { for filename }
       codes = set of 0..12 ;          { codes for irepr }

  var {  V A R I A B L E   D E C L A R A T I O N S   A N D   K E Y  }

diskfile : text ;                      { disk file variable }

new : boolean ;                        { true if new tgt base being created }

repaircodes : codes ;                  { repair codes for irepr }

crit : array[1..112,1..2] of real ;    { critical takeoff, taxi distances }

itgt : array[1..112,1..3] of integer ; { target category, hardness code, }
                                       {   and target group              }

tgt : array[1..112,1..5] of real ;     { coords, axis, & dimensions of target}

fname, backup : str14 ; { for file names }

ichc,          { identifies desired choice }
irepr,         { code for repair priorities }
lv,            { number of taxi surfaces }
mxptch,        { number of runway patches available }
narea,         { flag: 0 => compute TOL damage, 1 => don't compute }
nbldg,         { number of buildings }
ncp,           { number of takeoff/landing capable surfaces }
nelt,          { total number of targets }
nhard,         { number of hardness levels (total) }
nhardp,        { number of hardness levels for surfaces }
npprcw,        { min taxi width required, 0.0 suppresses search for taxiways }
ntgps          { number of distinct target groups }
  : integer ;

{  variables local to procedures:

  integers :
iopt           identifies desired option
i, j           loop counters
jchc           identifies desired choice
nmbr           desired target to delete, add or insert before
p              used for string manipulations ( for filenames )

  reals:
x1, x2, y1, y2   coordinate conversion variables                         }
```

```
{===================================================================}
procedure realread( var r : real ) ;
{ Procedure to read a real value with error checking. }
begin
  repeat {$I-} readln( r ) {$I+} until ( ioresult = 0 ) ;
end ;  { procedure realread }

{===================================================================}
procedure intread( var i : integer ) ;
{ Procedure to read an integer value with error checking. }
begin
  repeat {$I-} readln( i ) {$I+} until ( ioresult = 0 ) ;
end ;  { procedure intread }

{===================================================================}
procedure init ;
{ This procedure initializes AAPTGT variables. }
  var i : integer ;
begin
  repaircodes := [ 0..2, 10..12 ] ;  { acceptable values for irepr }
  narea := 0 ;
  nbldg := 0 ;
  ncp := 0 ;
  lv := 0 ;
  nelt := 0 ;
  ntgps := 0 ;
  for i := 1 to 112 do begin
    crit[i,1] := 0 ;
    crit[i,2] := 0 ;
  end ;
end ;  {  procedure init  }

{===================================================================}
procedure pause ;
{ Delays crt output until a key is pressed }
begin
  writeln ;
  write( '        p r e s s   a n y   k e y   t o   c o n t i n u e  ');
  repeat until keypressed ;
  clrscr ;
end ;  {  procedure pause  }
```

```
{===========================================================================}
procedure hardness ;
{ print hardness information }
  var i : integer ;
begin
  writeln ;
  writeln( 'Hardness Code     Target Type' ) ;
  writeln( '=============     ===========' ) ;
  for i := 1 to nhardp do writeln( i:7, '              pavement' ) ;
  for i := nhardp + 1 to nhard do writeln( i:7, '           non-pavement' ) ;
  writeln ;
end ; {  procedure hardness  }

{===========================================================================}
procedure tgtmatrix ;
{ print target matrix data }
  var i, j : integer ;
begin
  clrscr ;
  for i := 1 to nelt do begin
    if i mod 10 = 1 then begin
      writeln ;
      writeln( 'TGT     X       Y     AXIS                  TGT  HARD',
               '  TGT    CRIT DISTANCES' ) ;
      writeln( ' #    COORD   COORD   (DEG)  LENGTH   WIDTH  TYPE  CODE',
               ' GROUP  LENGTH   WIDTH' ) ;
      writeln( '---   -----   -----   -----  ------   -----  ----  ----',
               ' -----  ------   -----' ) ;
    end ;
    write( i:3 ) ;
    for j := 1 to 5 do write( tgt[i,j]:8:1 ) ;
    if itgt[i,1] = 0
      then write( '  BLDG' )
      else if crit[i,1] < 1
        then write( '   THY' )
        else write( '   TOL' ) ;
    write( itgt[i,2]:5, itgt[i,3]:6 ) ;
    if itgt[i,1] = 1 then write( crit[i,1]:10:1, crit[i,2]:9:1 ) ;
    writeln ;
    if i mod 10 = 0 then pause ;
  end ;
  if i mod 10 <> 0 then pause ;
end ;  { procedure tgtmatrix }
```

```
{==============================================================================}
procedure query( var ichc: integer; ix : integer ) ;
{ This procedure returns responses to several categories of
  yes-no and multiple choice questions. }
  procedure yn ;
  { Prompts for yes or no with "  (y/n)  ==)  "
    Sets ichc based on response:  y =) 0, n =) 1, (return) =) 0 }
    var ans : char ;
  begin
    write( '  (y/n)  ==)  ' ) ;
    repeat readln( ans ) until ( upcase( ans ) in [ 'Y','N' ] ) or eoln ;
    writeln ;
    if eoln or ( upcase( ans ) = 'N' )
      then ichc := 1
      else ichc := 0 ;
  end ; {  procedure yn  }

begin  {  body of procedure query  }
  ichc := 5 ;  { initialize out of range }

  case ix of

  1 : yn ;

  2 : begin
        writeln ;
        write( 'Do you want to update this value?' ) ;
        yn ;
      end ;

  3 : begin
        repeat
          writeln ;
          writeln( 'The number of targets currently defined = ', nelt,
                   '.' ) ;
          writeln ;
          if nelt ) 0
            then begin
              writeln ;
              writeln( 'Target database editing options.' ) ;
              writeln ;
              writeln( '                  0:  Delete a target.' ) ;
              writeln( '                  1:  Add a target.  ( Maximum ',
                       '112 )' ) ;
              writeln( '                  2:  Insert a target.' ) ;
              writeln( '                  3:  Review the target matrix.' ) ;
              writeln( '                  4:  Finish & save the new ',
                       'database.' ) ;
              write( 'Enter choice ==)  ' ) ;
              intread( ichc ) ;
              clrscr ;
            end
  .pa
```

```
          else begin
            writeln( 'You now must add at least one target.' ) ;
            ichc := 1 ;
          end ;
      until ichc in [ 0..4 ] ;
      writeln ;
    end ;

  4 : begin
      repeat
        writeln( 'There are two ways to enter the target coordinates.' ) ;
        writeln ;
        writeln( '                  0:  Target center coordinates.' ) ;
        writeln( '                  1:  End-midpoint coordinates.' ) ;
        write( 'Enter choice ==)  ' ) ;
        intread( ichc ) ;
        clrscr ;
        writeln ;
      until ichc in [ 0..1 ] ;
    end ;

  5 : begin
      repeat
        writeln ;
        writeln( 'Your options for this program are:' ) ;
        writeln ;
        writeln( '                   0:  Create a new target base.' ) ;
        writeln( '                   1:  Update an existing target ',
                 'base.' ) ;
        writeln ;
        write( 'Enter choice:  ==)  ' ) ;
        intread( ichc ) ;
      until ichc in [ 0..1 ] ;
      writeln ;
    end ;

  6 : begin
      write( 'Would you like a review of input prompts and options?' ) ;
      yn ;
      clrscr ;
      writeln ;
    end ;

  end ;  { case  }

end ;  {  procedure query  }
```

```
{==========================================================================}
procedure info( ix : integer ) ;
{ This procedure declutters the main program and procedures
  by placing text in a separate location. }
begin
  case ix of

  1 : begin
        clrscr ;
        writeln ;
        writeln( '            A A P T G T   T A R G E T   P R O G R A M' ) ;
        writeln( '            = = = = = =   = = = = = =   = = = = = = =' ) ;
        writeln ;
        writeln ;
        writeln( 'This program creates a target base for the modified ',
                 'version of the Attack' ) ;
        writeln ;
        writeln( 'Assessment Program - AAPMOD.  You must also ',
                 'generate a weapons base with' ) ;
        writeln ;
        writeln( 'AAPWPN and a laundered input file for AAPMOD ',
                 'using AAPMSN.' ) ;
        writeln ;
        writeln ;
        writeln( 'Keep track of the file names as you generate them.' ) ;
        writeln ;
      end ;

  2 : begin
        writeln ;
        writeln( 'Target definition' ) ;
        writeln ;
        writeln( 'Enter runways/TOL surfaces first, followed by taxiways, ',
                 'and then buildings.' ) ;
        writeln ;
        writeln( 'Remember that  AAPMOD  permits a maximum of 3 runways,  ',
                 '30 runway/taxiways, and' ) ;
        writeln( '112 runway/taxiway/buildings.' ) ;
        writeln ;
        writeln ;
        writeln( 'You must define each target in turn.' ) ;
        writeln ;
        writeln( 'Establish an x-y coordinate system for the complex.' ) ;
        writeln ;
        writeln( 'The positive x-axis becomes 0 degrees azimuth.' ) ;
        writeln ;
        writeln( 'Recommendation:  Let  the  main  runway  center  and  ',
                 'orientation  define  the' ) ;
        writeln( 'coordinate system.   Attack passes will also have ',
                 'aimpoints and azimuth defined' ) ;
        writeln( 'in this system.' ) ;
        pause ;
        writeln ;
```

```
end ;

3 : begin
    clrscr ;
    writeln ;
    writeln( 'Input prompts are defined as follows:' ) ;
    writeln ;
    writeln( '  Input options: ' ) ;
    writeln ;
    writeln( '              Target center coordinates.' ) ;
    writeln( '              End-midpoint  coordinates.' ) ;
    writeln ;
    writeln( 'For target center coordinates:' ) ;
    writeln ;
    writeln( '  X-coordinate:  X-coordinate  of the center of the ',
            'target  referenced  to  the' ) ;
    writeln( '              coordinate system of the complex.' ) ;
    writeln ;
    writeln( '  Y-coordinate:  Similar to x-coordinate.' ) ;
    writeln ;
    writeln( '        Axis:  Axis  orientation   is   target   ',
            'centerline   measured   CCW' ) ;
    writeln( '              ( in integer degrees )  from  +x-axis  of ',
            'coordinate system of' ) ;
    writeln( '              the complex.' ) ;
    writeln ;
    writeln( '       Length:  Length and width are self-explanatory.' ) ;
    writeln ;
    writeln( '        Width:  Width must be smaller than or equal to ',
            'length.' ) ;
    pause ;
    writeln ;
    writeln ;
    writeln( 'For end-midpoint coordinates:' ) ;
    writeln ;
    writeln( '  X-coordinate ( left-most short dimension ):   ',
            'X-coordinate of midpoint of the' ) ;
    writeln( '    end of the target with smaller x-value.' ) ;
    writeln ;
    writeln( '  X-coordinate, other end:   X-coord of midpoint of ',
            'opposite end of target.' ) ;
    writeln ;
    writeln( '  Y-coordinate, ( left-most short dimension ):  ',
            'Y-coordinate of midpoint of the' ) ;
    writeln( '    end of the target with smaller x-value.   ( If  ',
            'x1 = x2,  pick the end with' ) ;
    writeln( '    the smaller y. )' ) ;
    writeln ;
    writeln( '  Y-coordinate, other end:  Y-coord of midpoint of opposite',
            ' end of target.' ) ;
    pause ;
    writeln ;
    writeln( 'More prompts:' ) ;
```

```
      writeln ;
      writeln( '  Type of target:  0 = non-pavement ( building ).');
      writeln( '                   1 = pavement ( taxi or TOL ).') ;
      writeln ;
      writeln( '       Hardness:  Hardness of target ( for crater table ',
               'lookup ).' );
      writeln ;
      writeln( '   Target group:  Target group of the target in ',
               'question.' ) ;
      writeln ;
      writeln( ' Minimum length:  Minimum  clear  length  required for ',
               'TOL operations.  Enter' ) ;
      writeln( '                  zero for taxi-only pavements.' ) ;
      writeln ;
      writeln( '  Minimum width:  Minimum clear width required for:' ) ;
      writeln ;
      writeln( '                    Taxi:  if minimum length = 0 ',
               '( taxi-only ).' ) ;
      writeln( '                    Takeoff/Land:  if minimum length > 0.');
      writeln ;
      writeln( 'First, enter TOL capable pavements,  followed by  ',
               'taxi-only  capable pavements,' ) ;
      writeln( 'followed by buildings  or  non-pavements.   The same ',
               'hardness in a pavement and' ) ;
      writeln( 'a building requires two different code numbers.' ) ;
      pause ;
    end ;

  4 : begin
      clrscr ;
      writeln ;
      writeln ;
      writeln( 'Select crater repair priority:' ) ;
      writeln ;
      writeln ;
      writeln( '   0:  All TOL strips in order of target number.' ) ;
      writeln ;
      writeln( '   1:  Easiest TOL strip first, rest in order.' ) ;
      writeln ;
      writeln( '   2:  Repair only the easiest TOL strip.' ) ;
      writeln ;
      writeln( '  10:  All pavements in order of target number.' ) ;
      writeln ;
      writeln( '  11:  All approaches and easiest TOL strip first,' ) ;
      writeln( '         followed by others in target order.' ) ;
      writeln ;
      writeln( '  12:  All approaches and only the easiest TOL strip.' ) ;
      writeln ;
      writeln ;
    end ;
```

```
  5 : begin
       writeln( 'E R R O R :' ) ;
       writeln( 'You must enter runways first, taxiways second, and ',
                'non-pavements last.' ) ;
       writeln ;
     end ;

  end ;  {  case  }

end ;  {  procedure info  }

{==========================================================================}
function chkgood( nval, hi, lo : integer ) : boolean ;
{ This function returns false when nval is out of range. }
begin
  if ( nval < lo ) or ( nval > hi)
    then begin
       writeln ;
       writeln( 'Value must be between ',lo,' and ',hi ) ;
       writeln ;
       chkgood := false ;
     end
    else chkgood := true ;
end ; {  function chkgood  }

{==========================================================================}
procedure processfile( var fname : str14; which : str14 ) ;
{ Initial processing of file name entered by user. }
  var dot, goodname : boolean ;
      p, size, dotpos : integer ;
begin
  clrscr ;
  writeln ;
  writeln( 'For the following entries, allowable filenames may contain only ',
          'capital letters' ) ;
  writeln( 'and numbers.  If you enter lower case letters, they will be ',
         'treated as if they' ) ;
  writeln( 'are upper case.  The format must be as follows:' ) ;
  writeln ;
  writeln( '          D:XXXXXXXX.XXX' ) ;
  writeln ;
  writeln( '  where   D: is an optional drive specifier,' ) ;
  writeln( '          X is a digit ( 0 - 9 ) or a letter ( A - Z ),' ) ;
  writeln( '          XXXXXXXX is a filename 1 to 8 characters long, and' ) ;
  writeln( '          .XXX is an optional 1 to 3 character file type',
          ' ( not .BAK ).' ) ;
  writeln ;
  writeln( 'Examples of valid filenames:  MSN   MSN.1   AAPMOD.DTA    ',
          'B:1JAN84.AAP' ) ;
  writeln ;
  writeln ;
  writeln( 'If an existing file you need to use contains  characters  not ',
          'compatible with' ) ;
```

```
writeln( 'the above format,  you will need to ABORT this program by ',
        'pressing control-C,' ) ;
writeln( 'and then rename the file to make it compatible.' ) ;
pause ;
repeat
  goodname := true ;
  dot := false ;
  repeat
    writeln ;
    write( 'Enter name for the ', which, ' File  ==>  ' ) ;
    readln( fname ) ;
    writeln ;
    for p := 1 to 14 do fname[p] := upcase( fname[p] ) ;
    repeat                                  { delete blanks from }
      p := pos( ' ', fname ) ;              { filename           }
      if p <> 0 then delete( fname, p, 1 ) ;
    until p = 0 ;
  until fname <> '' ;             { trap for carriage return/blank filename}
  dotpos := pos( '.', fname ) ;   { filetype can only be 3 characters long }
  if dotpos <> 0 then delete( fname, dotpos + 4, 14 ) ;
  size := length( fname ) ;
  p := pos( ':', fname ) ;
  if ( p = 2 )
    then begin

        if ( not ( dotpos in [ 0, 4..11 ] ) )
                        or
           ( not ( fname[1] in [ 'A'..'Z' ] ) )
                        or
                ( size < 3 )
                        or
           ( ( dotpos = 0 ) and ( size > 10 ) )
                       then
                goodname := false ;

        if goodname then for p := 3 to size do begin
          if not ( fname[p] in [ '0'..'9', 'A'..'Z', '.' ] )
            then goodname := false ;
          if dot and ( fname[p] = '.' ) then goodname := false ;
          if fname[p] = '.' then dot := true ;
        end ;
      end
    else begin

        if ( not ( dotpos in [ 0, 2..9 ] ) )
                        or
                ( size < 0 )
                        or
           ( ( dotpos = 0 ) and ( size > 8 ) )
                       then
                goodname := false ;
```

```
        if goodname then for p := 1 to length( fname ) do begin
          if not ( fname[p] in [ '0'..'9', 'A'..'Z', '.' ] )
            then goodname := false ;
          if dot and ( fname[p] = '.' ) then goodname := false ;
          if fname[p] = '.' then dot := true ;
        end ;
      end ;
    if pos( '.BAK', fname ) <> 0 then begin
      goodname := false ;
      writeln( 'You may not enter a filename with the filetype ".BAK" .' ) ;
      writeln ;
      writeln( 'To use the backup .BAK file, rename it before executing ',
             'this program.' ) ;
      writeln ;
    end ;
    writeln ;
    if goodname
      then begin
        write( 'Confirm the correct filename is ( ', fname, ' ) ?' ) ;
        query( p, 1 ) ;
        if p = 1 then goodname := false ;
      end
      else begin
        writeln( 'Invalid filename ( ',fname, ' )' ) ;
        writeln ;
        writeln( 'Please try again, or press control-C to ABORT this ',
               'program.' ) ;
      end ;
  until goodname ;
end ;  { procedure processfile }

{=================================================================}
function filexists( fname : str14 ) : boolean ;
{ Checks to see if file fname exists on disk.  Returns true if it does. }
  var exists : boolean ;
begin
  assign( diskfile, fname ) ;
  {$I-} reset( diskfile ) ; {$I+  this checks if the file exists}
  exists := ioresult = 0 ;
  if not exists then begin
    writeln( 'File ( ', fname, ' ) does not exist.  Be sure to specify the ',
           'correct drive ' ) ;
    writeln( 'and the correct file name.' ) ;
    writeln ;
    writeln( '                To ABORT this program, press control-C.' ) ;
    pause ;
  end ;
  filexists := exists ;
end ;  { function filexists }
```

```
{==========================================================================}
procedure newval(var nvrbl : integer; ichc : integer ) ;
{ This procedure prompts for a new value and reads it. }
begin
  if ichc = 0 then begin
    write( 'Enter new value ==>  ' ) ;
    intread( nvrbl ) ;
    writeln ;
  end ;
end ;  {  procedure newval  }

{==========================================================================}
procedure pack( nmbr : integer ) ;
{ This procedure removes empty lines from the target array. }
  var i, j : integer ;
begin
  if nmbr <= nelt then begin
    for i := nmbr to nelt do begin
        for j :=1 to 5 do tgt[i,j] := tgt[i + 1, j] ;
        for j := 1 to 3 do itgt[i,j] := itgt[i + 1, j] ;
        crit[i,1] := crit[i + 1, 1] ;
        crit[i,2] := crit[i + 1, 2] ;
    end ;
    crit[nelt + 1, 1] := 0 ;
    crit[nelt + 1, 2] := 0 ;
  end ;
end ;  {  procedure pack  }

{==========================================================================}
procedure entgt( i : integer ) ;
{ This procedure enters targets in the target base array and
  updates the various counters. }
  var noerror : boolean ;
      ichc, iopt : integer ;
      x1, x2, y1, y2 : real ;

  procedure wideflag ;  { checks for wide targets to set the narea flag }
  begin
    if narea = 0 then
      if ( itgt[i,1] = 1 ) and ( tgt[i,5] > 899 ) then begin
        clrscr ;
        writeln ;
        writeln( 'Because the width of the target is so large ( greater ',
                 'than 899 ), a flag will' ) ;
        writeln( 'be set to suppress AAPMOD execution of routine OVLAP.   ',
                 'This routine searches' ) ;
        writeln( 'for overlapping areas of craters, and would take ',
                 'excessive execution time for' ) ;
        writeln( 'large width targets.  Note that the search is ',
                 'suppressed for ALL targets with' ) ;
        writeln( 'this flag set.' ) ;
        narea := 1 ;
      end ;
```

```
  end ;  { procedure wideflag }

begin { entgt }

  query( ichc, 6 ) ;
  if ichc = 0  then info( 3 ) ;
  if nhard = nhardp then begin
    writeln( 'Non-pavements cannot be entered as targets because all ',
             'hardness codes are' ) ;
    writeln( 'dedicated to pavements.  Therefore, Target ', i, ' is a ',
             'pavement.' ) ;
    writeln ;
    itgt[i,1] := 1 ;
  end ;
  if nhardp = 0 then begin
    writeln( 'Pavements cannot be entered as targets because all hardness ',
             'codes are' ) ;
    writeln( 'dedicated to non-pavements.  Therefore, Target ', i, ' is a ',
             'non-pavement.' ) ;
    itgt[i,1] := 0 ;
  end ;
  if ( nhard <> nhardp ) and ( nhardp <> 0 ) then
  repeat
    write( 'Type of target:  0 = bldg, 1 = pavement.  Enter code ==>  ' ) ;
    intread( itgt[i,1] ) ;
    writeln ;
  until chkgood( itgt[i,1], 1, 0 ) ;
  noerror := true ;
  if itgt[i,1] = 0  { target is non-pavement }
    then if i <= ncp + lv
      then begin
          info( 5 ) ;
          pack( i ) ;
          noerror := false ;
        end
      else nbldg := nbldg + 1
    else  { target is a pavement }
         if ( ( nbldg > 0 ) and ( i > ncp + lv + 1 ) ) or ( ncp + lv >= 30 )
           then begin
               if ( nbldg > 0 ) and ( i > ncp + lv + 1 ) then info( 5 ) ;
               if ncp + lv >= 30 then begin
                 writeln( 'E R R O R :' ) ;
                 writeln( 'Number of pavements exceeds 30.  Removed.' ) ;
                 writeln ;
               end ;
               pack( i ) ;
               noerror := false ;
             end
           else begin
               writeln( 'Target ',i,' : pavement.  Enter minimum length for ',
                        'TOL operations.' ) ;
               write( '( 0 implies taxi only )  ==>  ' ) ;
               realread( crit[i,1] ) ;
```

```
                writeln ;
                if crit[i,1] < 1
                  then if i <= ncp
                      then begin
                          info( 5 ) ;
                          pack( i ) ;
                          noerror := false ;
                        end
                      else begin
                          lv := lv + 1 ;
                          writeln( 'Pavement ',i,' is for taxi only.' ) ;
                          write( 'Enter minimum width for taxi operations.  ',
                                 '==>  ' ) ;
                          realread( crit[i,2] ) ;
                        end
                  else if ncp >= 3
                    then begin
                        writeln( 'E R R O R :' ) ;
                        writeln( 'No more than 3 TOL surfaces permitted.  ',
                                 'Removed.' ) ;
                        writeln ;
                        pack( i ) ;
                        noerror := false ;
                      end
                    else if i > ncp + 1
                      then begin
                          info( 5 ) ;
                          pack( i ) ;
                          noerror := false ;
                        end
                      else begin
                          ncp := ncp + 1 ;
                          writeln( 'Pavement ',i,' supports TOL operations.' ) ;
                          write( 'Enter minimum width for TOL operations.  ',
                                 '==>  ' ) ;
                          realread( crit[i,2] ) ;
                        end ;
              end ; { target is a pavement }
  if noerror then begin
    nelt := nelt + 1 ;
    repeat
      noerror := true ;
      hardness ;
      write( 'Enter hardness code ==>  ' ) ;
      intread( itgt[i,2] ) ;
      if ( ( itgt[i,2] > nhardp ) and ( itgt[i,1] = 1 ) )
                          or
        ( ( itgt[i,2] <= nhardp ) and ( itgt[i,1] = 0 ) )
       then begin
         noerror := false ;
         clrscr ;
         writeln ;
         writeln( 'E R R O R :' ) ;
```

```
        writeln ;
        writeln( 'Discrepancy in number of hardness codes, this target ' ) ;
        writeln( 'type and this particular code.' ) ;
        writeln ;
        writeln( 'Ensure that target element types and declared ',
                 'hardnesses agree.' ) ;
      end ;
   until noerror and chkgood( itgt[i,2], nhard, 1 ) ;
   clrscr ;
   writeln ;
   repeat
     write( 'Enter target group ==>  ' ) ;
     intread( itgt[i,3] ) ;
     writeln ;
   until chkgood( itgt[i,3], 15, 1 ) ;
   if itgt[i,3] > ntgps then ntgps := itgt[i,3] ;
   query( iopt, 4 ) ;
   if iopt = 0
     then begin
        writeln( 'Target center coordinates: ' ) ;
        writeln ;
        write( 'x-coordinate ==>  ' ) ;
        realread( tgt[i,1] ) ;
        writeln ;
        write( 'y-coordinate ==>  ' ) ;
        realread( tgt[i,2] ) ;
        writeln ;
        write( 'axis ( degrees ) ==>  ' ) ;
        realread( tgt[i,3] ) ;
        if tgt[i,3] >= 180 then tgt[i,3] := tgt[i,3] - 180 ;
        clrscr ;
        writeln ;
        repeat
          write( 'length ( larger or equal to width ) ==>  ' ) ;
          realread( tgt[i,4] ) ;
          writeln ;
          write( 'width ==>  ' ) ;
          realread( tgt[i,5] ) ;
          writeln ;
          if tgt[i,5] > tgt[i,4] then begin
            clrscr ;
            writeln ;
            writeln( 'E R R O R :' ) ;
            writeln( 'The width is greater than the length.  Reenter.' ) ;
            writeln ;
          end ;
        until tgt[i,5] <= tgt[i,4] ;
        wideflag ;
      end
```

```
else begin
   repeat
     write( 'x-coordinate ( left-most short dimension ) ==>  ' ) ;
     realread( x1 ) ;
     writeln ;
     write( 'x-coordinate, other end ==>  ' ) ;
     realread( x2 ) ;
     writeln ;
     write( 'y-coordinate, ( left-most short dimension ) ==>  ' ) ;
     realread( y1 ) ;
     writeln ;
     write( 'y-coordinate, other end ==>  ' ) ;
     realread( y2 ) ;
     tgt[i,4] := sqrt( sqr( x2 - x1 ) + sqr( y2 - y1) ) ;
     clrscr ;
     writeln ;
     write( 'width ( specify shortest dimension ) ==>  ' ) ;
     realread( tgt[i,5] ) ;
     writeln ;
     if tgt[i,5] > tgt[i,4] then begin
       clrscr ;
       writeln ;
       writeln( 'E R R O R :' ) ;
       writeln( 'The width of the target is greater than the ',
                'calculated length.  Be sure you' ) ;
       writeln( 'enter the target correctly.' ) ;
       writeln ;
     end ;
   until tgt[i,5] <= tgt[i,4] ;
   wideflag ;
   tgt[i,1] := ( x1 + x2 ) / 2 ;
   tgt[i,2] := ( y1 + y2 ) / 2 ;
   if x1 = x2
     then tgt[i,3] := 90
     else tgt[i,3] := arctan( ( y2 - y1 ) / ( x2 - x1 ) ) * 180 / pi ;
  end ;
 end ;
end ;  { procedure entgt  }
```

```
{==============================================================================}
procedure newtgt ;
{ This procedure creates a new target base to be written to a user defined
    file name. }
  var badname : boolean ;
      ichc : integer ;
begin
  repeat
    clrscr ;
    processfile ( fname, 'New Target' ) ;
    assign( diskfile, fname ) ;
    {$I-} reset( diskfile ) ;   {$I+ check to see if diskfile exists }
    badname := ioresult = 0 ;
    if badname then begin
      write( 'File < ', fname, ' > already exists.  Erase?' ) ;
      query( ichc, 1 ) ;
      if ichc = 0 then begin
        writeln ;
        write( 'Are you SURE you want to erase < ',fname,' >?' ) ;
        query( ichc, 1 ) ;
        if ichc = 0 then begin
          erase( diskfile ) ;
          badname := false ;
        end ;
      end ;
      writeln ;
    end ;
  until not badname ;
  clrscr ;
  info( 2 ) ;
  writeln( 'A value of 0 for the next input will suppress AAPMOD search ',
           'for taxi' ) ;
  writeln( 'approach to a clear strip.' ) ;
  writeln ;
  write( 'Enter minimum width required for taxi operations ==>  ' ) ;
  intread( opprcw ) ;
  writeln ;
  writeln ;
  writeln( 'Enter number of different hardness codes.   Note:  the same ',
           'absolute hardness ' ) ;
  writeln( 'level in a pavement  and again in a building will require two ',
           'different codes.' ) ;
  repeat
    writeln ;
    write( 'Number of hardness levels (max 11) ==>  ' ) ;
    intread( nhard ) ;
  until chkgood( nhard, 11, 1 ) ;
  writeln ;
```

```
    repeat
      writeln ;
      writeln( 'Enter number of hardness codes associated with pavements.' ) ;
      write( '( Max is ',nhard,' ) ==>  ' ) ;
      intread( nhardp ) ;
    until chkgood( nhardp, nhard, 0 ) ;
    clrscr ;
    hardness ;
    repeat
      if nelt < 112
        then begin
            entgt( nelt + 1 ) ;
            write( 'Do you want to quit now?' ) ;
            query( ichc, 1 ) ;
          end
        else begin
            writeln( '112 targets already entered. You will have to delete ',
                     'targets before more can' ) ;
            writeln( 'be entered.  Complete your other entries, exit and ',
                     're-enter AAPTGT to delete' ) ;
            writeln( 'and add targets.' ) ;
            ichc := 0 ;
          end ;
    until ichc = 0 ;
    writeln ;
    writeln( 'Target complex repair capability:' ) ;
    writeln ;
    write( 'How many patches will resources allow ?  ==>  ' ) ;
    intread( mxptch ) ;
    writeln ;
    repeat
      info( 4 ) ;
      write( 'Choice ==>  ' ) ;
      intread( irepr ) ;
    until irepr in repaircodes ;
    writeln ;
  end ; {  procedure newtgt  }
```

```
{==========================================================================}
procedure deltgt( nmbr : integer ) ;
{ This procedure deletes a target entry and repacks the array. }
  var i : integer ;
      widthok : boolean ;
begin
  if nmbr <= ncp
    then ncp := ncp - 1
    else if nmbr <= ncp + lv
           then lv := lv - 1
           else nbldg := nbldg - 1 ;
  nelt := nelt - 1 ;
  pack( nmbr ) ;
  if narea = 1 then begin
    widthok := true ;
    for i := 1 to ncp + lv do if tgt[i,5] > 899 then widthok := false ;
    if widthok then begin
      narea := 0 ;
      writeln ;
      writeln( 'The flag to suppress AAPMOD routine OVLAP is now reset to ',
               'enable search.' ) ;
      writeln ;
    end ;
  end ;
end ; {  procedure deltgt  }

{=========================================================================}
procedure instgt( nmbr : integer ) ;
{ This procedure inserts a new entry above a specified target
  by moving the rest of the array down and inserting. }
  var i, j : integer ;
begin
  for i := nelt downto nmbr do begin
    for j := 1 to 5 do tgt[i + 1, j ] := tgt[i,j] ;
    for j := 1 to 3 do itgt[i + 1, j] := itgt[i,j] ;
    crit[i + 1, 1] := crit[i,1] ;
    crit[i + 1, 2] := crit[i,2] ;
  end ;
  entgt( nmbr ) ;
end ; {  procedure instgt  }
```

```
{==================================================================}
procedure oldtgt ;
{ This procedure updates an existing target base compatible
  with the AAPMOD series of programs. }
  var badname : boolean ;
      i, ichc, j, jchc, nmbr, p : integer ;
begin
  repeat processfile( fname, 'Target' ) until filexists( fname ) ;
  writeln ;
  writeln( '        R e a d i n g   f i l e  <', fname, ' > . . .' ) ;
  writeln ;
  writeln ;
  read( diskfile, nelt, ntgps, npprcw, narea ) ;
  for i := 1 to nelt do begin
    for j := 1 to 5 do read( diskfile, tgt[i,j] ) ;
    for j := 1 to 3 do read( diskfile, itgt[i,j] ) ;
    if itgt[i,1] = 1 then read( diskfile, crit[i,1], crit[i,2] ) ;
  end ;
  read( diskfile, ncp, lv ) ;
  read( diskfile, nhard, nhardp ) ;
  read( diskfile, mxptch, irepr ) ;
  close( diskfile ) ;
  nbldg := nelt - ncp - lv ;
  writeln ;
  writeln( 'Current min width for taxi is ', npprcw, '.' ) ;
  if npprcw = 0
    then writeln( 'This value will suppress AAPMOD search for taxi approach ',
                  'to a clear strip.' ) ;
  query( ichc, 2 ) ;
  newval( npprcw, ichc ) ;
  repeat
    clrscr ;
    hardness ;
    write( 'Do you want to update this table?' ) ;
    query( jchc, 1 ) ;
    if jchc = 0 then begin
      writeln( 'Current total hardness levels (surfaces plus buildings) ',
               '= ',nhard ) ;
      query( ichc, 2 ) ;
      repeat
        newval( nhard, ichc ) ;
        ichc := 0 ;
      until chkgood( nhard, 11, 1 ) ;
      writeln( 'Current hardness levels for pavements = ',nhardp ) ;
      query(ichc,2) ;
      repeat
        newval( nhardp, ichc ) ;
        ichc := 0 ;
      until chkgood( nhardp, nhard, 0 ) ;
    end ;
  until jchc = 1 ;
  clrscr ;
```

```
repeat

  query( ichc, 3 ) ;

  case ichc of

  0 : if nelt = 0
        then writeln( 'There are no target elements left to delete.' )
        else begin
            repeat
              write( 'Which target element is to be deleted?  ==>  ' ) ;
              intread( nmbr ) ;
              writeln ;
            until chkgood( nmbr, nelt, 1 ) ;
            write( 'Are you SURE you want to delete target ',nmbr,' ?' ) ;
            query( jchc, 1 ) ;
            clrscr ;
            writeln ;
            if jchc = 0
              then begin
                  deltgt( nmbr ) ;
                  writeln( 'Target ', nmbr, ' deleted.' ) ;
                end
              else writeln( 'Target ', nmbr, ' NOT deleted.' ) ;
            writeln ;
          end ;

  1 : if nelt >= 112
        then begin
            writeln( 'There are 112 targets already defined.  You must ',
                     'delete a target before you can' ) ;
            writeln( 'add another one.' ) ;
          end
        else entgt( nelt + 1 ) ;

  2 : case nelt of   { case for target insert }

      0 : entgt( 1 ) ;

  1..111 : begin
            repeat
              writeln( 'The new target will be inserted at the position ',
                       'you specify, displacing' ) ;
              writeln( 'previous entries.' ) ;
              writeln ;
              writeln( 'You presently have ', ncp, ' TOL surfaces and ',
                       lv, ' taxi surfaces.' ) ;
              writeln ;
              write( 'New target to be inserted at position # ==>  ' ) ;
              intread( nmbr ) ;
              writeln ;
            until chkgood( nmbr, nelt + 1, 1 ) ;
```

```
            if nmbr = nelt + 1
              then entgt( nmbr )
              else instgt( nmbr ) ;
          end ;

      else begin
            writeln( 'There are 112 targets already defined.  You must ',
                     'delete a target before you can' ) ;
            writeln( 'insert another one.' ) ;
          end ;

      end ;  { case for target insert }

    3 : tgtmatrix ;

    end ;  { case }

  until ichc = 4 ;

  writeln( 'Current number of patches available is ', mxptch ) ;
  query( ichc, 2 ) ;
  newval( mxptch, ichc ) ;
  info( 4 ) ;
  writeln( 'Current crater repair priority is ',irepr, '.' ) ;
  writeln ;
  if irepr in repaircodes
    then query( ichc, 2 )
    else ichc := 0 ;
  repeat
    newval( irepr, ichc ) ;
  until irepr in repaircodes ;
  backup := fname ;                           { create a filename equal to  }
  p := pos( '.',backup ) ;                    { specified filename except   }
  if p <> 0 then delete( backup, p, 4 ) ;     { with a .BAK filetype.       }
  backup := backup + '.BAK' ;
  assign( diskfile, backup ) ;
  {$I-} reset( diskfile ) ; {$I+}             { see if old .BAK file exists }
  if ioresult = 0 then erase ( diskfile ) ;   { erase old .BAK file }
  assign ( diskfile, fname ) ;
  rename( diskfile, backup ) ;      { old file now saved with .BAK extension }
end ;  { procedure oldtgt }
```

```
{=============================================================}
procedure outdat ;
{ This procedure writes the target base to a new file with the name
  specified by the user at program invocation.  If the file specified
  was an existing file named <filename>.<filetype>, it was renamed
  <filename.BAK> by procedure oldbas and saved. }
  var i, j : integer ;
begin
  clrscr ;
  writeln ;
  writeln( 'D A T A   I N P U T   C O M P L E T E' ) ;
  writeln ;
  writeln( 'Be sure to write down hardness information.  You will need this ',
           'when forming ' ) ;
  writeln( 'the weapons base in program AAPWPN.' ) ;
  writeln ;
  writeln( 'Total number of hardness codes = ', nhard, '.' ) ;
  hardness ;
  pause ;
  writeln ;
  writeln( 'You are now finished developing the target base.  The next step ',
           'is to define a' ) ;
  writeln( 'weapons base with AAPWPN.'  ) ;
  writeln ;
  writeln ;
  if not new then writeln( 'Old file < ', fname, ' > will be renamed < ',
                           backup, ' > and saved.' ) ;
  writeln ;
  writeln ;
  writeln( 'W r i t i n g   t o   d i s k   f i l e   < ', fname, ' > ...' ) ;
  writeln ;
  assign( diskfile, fname ) ;                { reassign file to use as the }
  rewrite( diskfile ) ;                      { specified filename          }
  writeln( diskfile, nelt:5, ntgps:5, npprcw:5, narea:5 ) ;
  for i := 1 to nelt do begin
    for j := 1 to 5 do write( diskfile, tgt[i,j]:9:1 ) ;
    for j := 1 to 3 do write( diskfile, itgt[i,j]:5 ) ;
    writeln( diskfile ) ;
    if itgt[i,1] = 1 then writeln( diskfile, crit[i,1]:9:1, crit[i,2]:9:1 ) ;
  end ;
  writeln( diskfile, ncp:5, lv:5 ) ;
  writeln( diskfile, nhard:5, nhardp:5 ) ;
  writeln( diskfile, mxptch:5, irepr:5 ) ;
  close( diskfile ) ;
end ;  {  procedure outdat  }
```

```
{====================  MAIN PROGRAM  ====================}
begin

init ;
info( 1 ) ;
query( ichc, 5 ) ;
new := ichc = 0 ;
query( ichc, 6) ;
if ichc = 0 then info( 3 ) ;
if new then newtgt else oldtgt ;
outdat ;
writeln( 'E N D   O F   P R O G R A M' ) ;

end.
```

```
{==========================================================================}
{================ file AAPWPN.PAS     18 Feb 85 ============================}
{==========================================================================}

{$IAAPWPN1.PAS  compiler directive to include file AAPWPN1.PAS}
```

```
{==========================================================================}
{=================  FILE AAPWPN1.PAS    18 Feb 1985  ======================}
{==========================================================================}
program aapwpn ;

  type str14 = string[14] ;           { for filenames                }

  var  { V A R I A B L E   D E C L A R A T I O N S   A N D   K E Y  }

diskfile : text ;                    { disk file variable }

crtab : array[1..11,1..6,1..2] of real ;

{ crtab[i,j,k] - crater tabulated data
                 i - surface type (hardness code)
                 j - weapons type
                 k - crater radii subdivided by surface type
                      pavement: 1=) TOL crater, 2=) taxiway crater
                      building: 1=) near miss, 2=) direct hit        }

ipat : array[1..12,1..4] of integer ;

{ ipat[i,j] - for each defined pattern(i), j:
               1 - number of weapons delivered ( 12 or less )
               2 - number of submunitions per weapon
               3 - weapon code (crtab[i,j,k] index number j)
               4 - pattern shape
                    0 ==) General purpose bomb
                    1 ==) CBU rectangular footprint
                    2 ==) CBU elliptical footprint
                    3 ==) guided bomb                               }

patt : array[1..12,1..34] of real ;

{ patt[i,j] - pattern data: for each pattern(i), j:
               1-delivery or optimal guidance range error sigma
               2-delivery or optimal guidance deflection error sigma
               3-ballistic or near-miss range error sigma
               4-ballistic or near miss deflection error sigma
               5-CBU half pattern length or gross-error range error sigma
               6-CBU half pattern width or gross-error deflection sigma
               7-CBU half void length or probability of optimal guidance
               8-CBU half void width or prob of near-miss or better guidance
               9-weapon or cannister fuze reliability
               10-bomblet reliability
               11, 13, ... 33 - stick impact points ( range from aimpoint )
               12, 14, ... 34 - stick impact points ( offset from aimpoint ) }

fname, backup : str14 ;  { for file names }
```

```
ichc,                     { choice flag                                   }
nhard,                    { total hardness codes for bldgs + surfaces     }
nhardp,                   { hardness codes for surfaces                   }
npatt,                    { number of patterns ( 12 or less )             }
nsqr,                     { crater input flag; 0 => square, 1 => round    }
nwpn                      { # of different weapon types ( 6 or less )     }
  : integer ;

{ integers local to various procedures:

i, j, 1j,                  loop counters
jchc                       choice flag
jdef                       deflection counter (index in patt matrix)
jrng                       range counter (index in pattern matrix)
np                         number of release pulses
opatt                      pattern option flag
p                          integer for filename (string) manipulations

  reals local to various procedures:

al,                        length of CBU pattern
aw,                        width of CBU pattern
cep1,                      optimal guidance (guided munitions)
cep2,                      near miss CEP (guided munitions)
df,                        distance between bomb impacts
dt,                        delta t or intervalometer setting
fps,                       true airspeed in feet per second
ft,                        width of stick/string of bombs
sstart,                    starting location of stick
stickl,                    stick length in feet
tas,                       true airspeed in knots
theta,                     dive angle
vl,                        length of CBU void
vw,                        width of CBU void
wft                        stick half-width                                 }

{=========================================================================}
procedure init ;  { zeroes out all arrays }
  var i, j : integer ;
begin
  for i := 1 to 11 do
    for j := 1 to 6 do begin
      crtab[i,j,1] := 0 ;
      crtab[i,j,2] := 0 ;
  end ;
  for i := 1 to 12 do begin
    for j := 1 to 4 do ipat[i,j] := 0 ;
    for j := 1 to 34 do patt[i,j] := 0 ;
  end ;
end ; { procedure init }
```

```
{==========================================================================}
procedure realread( var r : real ) ;
{ Procedure to read a real value with error checking. }
begin
  repeat {$I-} readln( r ) {$I+} until ( ioresult = 0 ) ;
  writeln ;
end ; { procedure realread }

{==========================================================================}
procedure intread( var i : integer ) ;
{ Procedure to read an integer value with error checking. }
begin
  repeat {$I-} readln( i ) {$I+} until ( ioresult = 0 ) ;
  writeln ;
end ; { procedure intread }

{==========================================================================}
procedure pause ;
{ Delays crt output until a key is pressed. }
begin
  writeln ;
  write( '          p r e s s   a n y   k e y   t o   c o n t i n u e   ') ;
  repeat until keypressed ;
  clrscr ;
end ; { procedure pause }

{==========================================================================}
procedure review ;
{  Display how many weapons and patterns are defined.  }
begin
  writeln ;
  write( 'You presently have ', nwpn, ' weapon' ) ;
  if nwpn <> 1 then write( 's' ) ;
  write( ' and ',npatt,' pattern' ) ;
  if npatt <> 1 then write( 's' ) ;
  writeln( ' defined.' ) ;
  writeln ;
  writeln ;
end ; { procedure review }
```

```
{============================================================================}
procedure query( var ichc: integer; ix : integer ) ;
{ This procedure returns responses to several categories of yes-no and
multiple choice questions.                                                    }
  var ans : char ;
begin
  ichc := -1 ;      { initialize ichc out of range }

  case ix of

1 : begin
      write( '  (y/n)  ==>  ' ) ;
      repeat readln( ans ) until ( upcase( ans ) in [ 'Y','N' ] ) or eoln ;
      writeln ;
      if eoln or ( upcase( ans ) = 'N' )
        then ichc := 1
        else ichc := 0 ;
    end ;

2 : repeat
      review ;
      if nwpn = 0
        then begin
            writeln( 'You now must add a weapon.' ) ;
            pause ;
            writeln ;
            ichc := 1 ;
          end
        else begin
            writeln( 'Weapon section editing options.' ) ;
            writeln ;
            writeln( '                    0:  Delete a weapon and its ',
                     'corresponding patterns.' ) ;
            writeln( '                    1:  Add a weapon ( maximum is 6 ).' ) ;
            writeln( '                    2:  Insert a weapon.' ) ;
            writeln( '                    3:  Redefine a weapon.' ) ;
            writeln( '                    4:  Review the weapon matrix.' ) ;
            writeln( '                    5:  Review the attack pattern ',
                     'matrix.' ) ;
            writeln( '                    6:  End weapon section editing.' ) ;
            write  ( 'Enter choice ==>  ' ) ;
            intread( ichc ) ;
            clrscr ;
            writeln ;
          end ;
    until ichc in [ 0..6 ] ;
```

```
3 : repeat
      review ;
      if npatt = 0
        then begin
            writeln( 'You must now add a pattern.' ) ;
            pause ;
            writeln ;
            ichc := 1 ;
          end
        else begin
            writeln( 'Attack pattern section editing options.' ) ;
            writeln ;
            writeln( '              0:  Delete a pattern.' ) ;
            writeln( '              1:  Add a pattern.' ) ;
            writeln( '              2:  Insert a pattern.' ) ;
            writeln( '              3:  Review the weapon matrix.' ) ;
            writeln( '              4:  Review the attack pattern ',
                     'matrix.' ) ;
            writeln( '              5:  Save data and end the program.' ) ;
            write  ( 'Enter choice ==>  ' ) ;
            intread( ichc ) ;
            clrscr ;
            writeln ;
          end ;
    until ichc in [ 0..5 ] ;

4 : repeat
      writeln ;
      writeln( 'Individual weapon trajectory code:' ) ;
      writeln ;
      writeln( '              0:  Single general purpose weapons.' ) ;
      writeln( '              1:  CBU, rectangular pattern ( voids not ',
               'allowed ).' ) ;
      writeln( '              2:  CBU, elliptical pattern ( voids ',
               'allowed ).' ) ;
      writeln( '              3:  Guided munition' ) ;
      write  ( 'Enter code ==>  ' ) ;
      intread( ichc ) ;
    until ichc in [ 0..3 ] ;

5 : repeat
      clrscr ;
      writeln ;
      writeln( 'You may choose to drop the string of bombs in singles or in ',
               'pairs.' ) ;
      writeln ;
      writeln( '              1:  Drop weapons in singles.' ) ;
      writeln( '              2:  Drop weapons in pairs.' ) ;
      write  ( 'Enter choice ==>  ' ) ;
      intread( ichc ) ;
    until ichc in [ 1, 2 ] ;
```

```
6 : repeat
      clrscr ;
      writeln ;
      writeln( 'You must now describe the weapons stick.  Manual entry ',
               'requires you to enter' ) ;
      writeln( 'each impact point by deflection and range from the aimpoint.',
               '  Alternatively,' ) ;
      writeln( 'the automatic calculation procedure requires you to enter ',
               'the  release  dive' ) ;
      writeln( 'angle, true airspeed, intervalometer setting, and width of ',
               'the stick.' ) ;
      writeln ;
      writeln( '                  0: Manual entry.' ) ;
      writeln( '                  1: Automatic calculation.' ) ;
      write( 'Enter choice ==>  ' ) ;
      intread( ichc ) ;
    until ichc in [ 0, 1 ] ;

7 : repeat
      clrscr ;
      writeln ;
      writeln( 'Describe error distribution for guided munitions with ',
               'standard deviation' ) ;
      writeln( '( sigma ) or range and deflection errors probable ',
               '( REP / DEP ):') ;
      writeln ;
      writeln( '                  1:  Entry as REP / DEP' ) ;
      writeln( '                  2:  Entry as standard deviation ',
               '( sigma )' ) ;
      write( 'Enter choice ==>  ' ) ;
      intread( ichc ) ;
    until ichc in [ 1, 2 ] ;

8 : repeat
      writeln ;
      writeln( 'AAPMOD will use square craters, but you may choose the input ',
               'mode:' ) ;
      writeln ;
      writeln( '                      0:  Half-length of side of square ',
               'crater.' ) ;
      writeln( '                      1:  Radius of circular crater.' ) ;
      write( 'Enter your choice ==>  ' ) ;
      intread( ichc ) ;
      clrscr ;
      writeln ;
    until ichc in [ 0, 1 ] ;

  end ; { case }

end ;  { procedure query }
```

```
{==========================================================================}
procedure hardness ;
{ Print hardness information. }
  var i : integer ;
begin
  writeln ;
  writeln ;
  writeln( 'Hardness Code     Target Type' ) ;
  writeln( '=============     ===========' ) ;
  for i := 1 to nhardp do writeln( i:7, '            pavement' ) ;
  for i := nhardp + 1 to nhard do writeln( i:7, '        non-pavement' ) ;
  writeln ;
end ; {  procedure hardness  }

{==========================================================================}
function chkgood( nval, hi, lo : integer ) : boolean ;
{ This function returns false when nval is out of range. }
begin
  if ( nval < lo ) or ( nval > hi)
    then begin
        writeln ;
        writeln( 'Value must be between ',lo,' and ',hi ) ;
        writeln ;
        chkgood := false ;
      end
    else chkgood := true ;
end ; { function chkgood }

{==========================================================================}
function probchk( prob : real ) : boolean ;
{ This function returns false when prob is out of range ( 0 to 1 ). }
begin
  if ( prob < 0 ) or ( prob > 1 )
    then begin
        writeln ;
        writeln( 'Enter a value between 0 and 1.' ) ;
        writeln ;
        probchk := false ;
      end
    else probchk := true ;
end ; { function probchk }
```

```
{==================================================================}
procedure processfile( var fname : str14; which : str14 ) ;
{ Initial processing of file name entered by user. }
  var dot, goodname : boolean ;
      p, size, dotpos : integer ;
begin
  clrscr ;
  writeln ;
  writeln( 'For the following entries, allowable filenames may contain only ',
           'capital letters' ) ;
  writeln( 'and numbers.  If you enter lower case letters, they will be ',
           'treated as if they' ) ;
  writeln( 'are upper case.  The format must be as follows:' ) ;
  writeln ;
  writeln( '          D:XXXXXXXX.XXX' ) ;
  writeln ;
  writeln( '   where   D: is an optional drive specifier,' ) ;
  writeln( '          X is a digit ( 0 - 9 ) or a letter ( A - Z ),' ) ;
  writeln( '          XXXXXXXX is a filename 1 to 8 characters long, and' ) ;
  writeln( '          .XXX is an optional 1 to 3 character file type',
           ' ( not .BAK ).' ) ;
  writeln ;
  writeln( 'Examples of valid filenames:  MSN    MSN.1    AAPMOD.DTA    ',
           'B:1JAN84.AAP' ) ;
  writeln ;
  writeln ;
  writeln( 'If an existing file you need to use contains  characters  not ',
           'compatible with' ) ;
  writeln( 'the above format,  you will need to ABORT this program by ',
           'pressing control-C,' ) ;
  writeln( 'and then rename the file to make it compatible.' ) ;
  pause ;
  repeat
    goodname := true ;
    dot := false ;
    repeat
      writeln ;
      write( 'Enter name for the ', which, ' File ==>  ' ) ;
      readln( fname ) ;
      writeln ;
      for p := 1 to 14 do fname[p] := upcase( fname[p] ) ;
      repeat                                  { delete blanks from }
        p := pos( ' ', fname ) ;              { filename           }
        if p <> 0 then delete( fname, p, 1 ) ;
      until p = 0 ;
    until fname <> '' ;             { trap for carriage return/blank filename}
    dotpos := pos( '.', fname ) ;    { filetype can only be 3 characters long }
    if dotpos <> 0 then delete( fname, dotpos + 4, 14 ) ;
    size := length( fname ) ;
    p := pos( ':', fname ) ;
```

```
if ( p = 2 )
  then begin

      if ( not ( dotpos in [ 0, 4..11 ] ) )
                       or
         ( not ( fname[1] in [ 'A'..'Z' ] ) )
                       or
                  ( size < 3 )
                       or
         ( ( dotpos = 0 ) and ( size > 10 ) )
                      then
                goodname := false ;

      if goodname then for p := 3 to size do begin
        if not ( fname[p] in [ '0'..'9', 'A'..'Z', '.' ] )
          then goodname := false ;
        if dot and ( fname[p] = '.' ) then goodname := false ;
        if fname[p] = '.' then dot := true ;
      end ;
    end
  else begin

      if ( not ( dotpos in [ 0, 2..9 ] ) )
                       or
                  ( size < 0 )
                       or
         ( ( dotpos = 0 ) and ( size > 8 ) )
                      then
                goodname := false ;

      if goodname then for p := 1 to length( fname ) do begin
        if not ( fname[p] in [ '0'..'9', 'A'..'Z', '.' ] )
          then goodname := false ;
        if dot and ( fname[p] = '.' ) then goodname := false ;
        if fname[p] = '.' then dot := true ;
      end ;
    end ;
if pos( '.BAK', fname ) <> 0 then begin
  goodname := false ;
  writeln( 'You may not enter a filename with the filetype ".BAK" .' ) ;
  writeln ;
  writeln( 'To use the backup .BAK file, rename it before executing ',
           'this program.' ) ;
  writeln ;
end ;
writeln ;
if goodname
  then begin
     write( 'Confirm the correct filename is ( ', fname, ' ) ?' ) ;
     query( p, 1 ) ;
     if p = 1 then goodname := false ;
    end
```

```
    else begin
      writeln( 'Invalid filename < ',fname, ' >' ) ;
      writeln ;
      writeln( 'Please try again, or press control-C to ABORT this ',
               'program.' ) ;
    end ;
  until goodname ;
end ;  { procedure processfile }

{==============================================================}
function filexists( fname : str14 ) : boolean ;
{ Checks to see if file fname exists on disk.  Returns true if it does. }
  var exists : boolean ;
begin
  assign( diskfile, fname ) ;
  {$I-} reset( diskfile ) ; {$I+  this checks if the file exists}
  exists := ioresult = 0 ;
  if not exists then begin
    writeln( 'File < ', fname, ' > does not exist.  Be sure to specify the ',
             'correct drive ' ) ;
    writeln( 'and the correct file name.' ) ;
    writeln ;
    writeln( '                To ABORT this program, press control-C.' ) ;
    pause ;
  end ;
  filexists := exists ;
end ;  { function filexists }

{==============================================================}
procedure info( ix : integer ) ;
{ This procedure declutters the main program and procedures by placing text in
a separate location. }
begin
  case ix of

1 : begin
      clrscr ;
      writeln ;
      writeln( '              A A P W P N   W E A P O N S   P R O G R A M' ) ;
      writeln( '              = = = = = =   = = = = = = =   = = = = = = =' ) ;
      writeln ;
      writeln ;
      writeln( 'This program  creates a weapons base  and an attack pattern  ',
               'base  for the' ) ;
      writeln ;
      writeln( 'modified version of the attack assessment program - AAMOD.',
               '  This data is ' ) ;
      writeln ;
      writeln( 'to be used in concert with a target base developed by the ',
               'program  AAPTGT.' ) ;
      writeln ;
      writeln( 'AAPTGT and AAPWPN are independent of one another and permit ',
               'easy update of ' ) ;
```

```
      writeln ;
      writeln( 'the target, weapon, and attack pattern data bases.  The ',
               'output file of ' ) ;
      writeln ;
      writeln( 'this program  and AAPTGT  are combined by the program  ',
               'AAPMSN  to form the' ) ;
      writeln ;
      writeln( 'laundered input file for the main program - AAPMOD.  Your ',
               'options for this' ) ;
      writeln ;
      writeln( 'program are:' ) ;
      writeln ;
      writeln( '                 0:  Create a new weapons/pattern base.' ) ;
      writeln( '                 1:  Update an existing weapons/pattern ',
               'base.' ) ;
      ichc := -1 ;
      repeat
        write  ( 'Enter choice ==>  ' ) ;
        intread( ichc ) ;
      until chkgood( ichc, 1, 0 ) ;

    end ;

2 : begin
      clrscr ;
      writeln ;
      writeln( 'Crater table format is:' ) ;
      writeln ;
      writeln( 'target        wpn 1        wpn 2        wpn 3    ...' ) ;
      writeln( 'hardness' ) ;
      writeln( '   1       crater size1  crater size1  crater size1...' ) ;
      writeln( '   1       crater size2  crater size2  crater size2...' ) ;
      writeln( '   2            .             .             .' ) ;
      writeln( '   2            .             .             .' ) ;
      writeln( '   3            .             .             .' ) ;
      writeln( '   3            .             .             .' ) ;
      writeln( '   .            .             .             .' ) ;
      writeln( '   .            .             .             .' ) ;
      writeln ;
      writeln( 'Crater  size1  is the size for the weapon against  TOL ',
               'surfaces if the hardness' ) ;
      writeln( 'code is for pavements,  or  near miss  crater size ',
               'for buildings.  Crater size2' ) ;
      writeln( 'is  the  size  for  the  weapon against taxiways  ',
               'if the  hardness  code is for' ) ;
      writeln( 'pavements, or direct hits against buildings.' ) ;
      writeln ;
      pause ;
    end ;
```

```
3 : begin
     clrscr ;
     writeln ;
     writeln( 'Crater data' ) ;
     writeln ;
     writeln( 'The damage mechanism of AAP/AAPMOD is cratering.' ) ;
     writeln ;
     writeln( 'AAPMOD uses a 3-d array ( i, j, k ) that you must now ',
              'generate.' ) ;
     writeln ;
     writeln( 'The size of a crater is defined by the combination of:' ) ;
     writeln ;
     writeln( '   i:  a defined level of target hardness, thickness, type ',
              'of material,  etc.,' ) ;
     writeln( '        all combined into one, categorical hardness code.' ) ;
     writeln ;
     writeln( '   j:  a defined type of warhead,  and' ) ;
     writeln ;
     writeln( '   k:  a defined type of interaction.' ) ;
     writeln ;
     pause ;
     writeln ;
     writeln( 'This section builds the crater table for each hardness code ',
              'you specified.  The' ) ;
     writeln( 'codes will be handled in the order entered in  AAPTGT.   ',
              'For each hardness code' ) ;
     writeln( '( pavement/building type ), you  will  enter  crater  sizes ',
              ' for  each  of  the' ) ;
     writeln( 'potential weapons  to  be  defined.   These weapons interact ',
              'in varying manners' ) ;
     writeln( 'with the  pavements/buildings  depending on how close to the ',
              'target the weapons' ) ;
     writeln( 'impact.  For a specific weapon and target,  you will ',
              'enter both of the possible' ) ;
     writeln( 'interaction types  and  move  to  the  next weapon.  For ',
              'pavements, interaction' ) ;
     writeln( 'types are:  crater size to deny takeoff/landings  ',
              'and  crater size to deny taxi' ) ;
     writeln( '( probably smaller ).  For buildings,  interaction ',
              'types are:   crater size for' ) ;
     writeln( 'near miss and crater size for direct hit  ',
              '( probably larger ).   After entering' ) ;
     writeln( 'all the weapons effects against a particular ',
              'target,  the program  will move to' ) ;
     writeln( 'the target in the sequence developed by AAPTGT.' ) ;
     pause ;
   end ;
```

```
4 : begin
      clrscr ;
      writeln ;
      writeln( '              )))) R E A D   C A R E F U L L Y  ',
              '((((' ) ;
      writeln ;
      writeln( 'For the combination of hardness and warhead, AAPMOD  uses ',
              'two different crater' ) ;
      writeln( 'sizes.' ) ;
      writeln ;
      writeln ;
      writeln( 'If the  target  hardness  was  assigned to a pavement,  ',
              'first enter the size of' ) ;
      writeln( 'disruption  severe enough  to deny  hi-speed  TOL ',
              'operations,  then the size of' ) ;
      writeln( 'disruption severe enough to deny taxi operations.' ) ;
      writeln ;
      writeln ;
      writeln( 'If the target was a building,  first enter the crater radius ',
              'resulting  from  a' ) ;
      writeln( 'near-miss, then the radius resulting from a direct hit.' ) ;
      writeln ;
    end ;

5 : begin
      writeln( 'The maximum number of weapons is 6.' ) ;
      writeln( 'You must delete a weapon prior to adding a new one.' ) ;
      writeln ;
    end ;

6 : begin
      writeln( 'The maximum number of attack patterns is 12.' ) ;
      writeln( 'You will have to delete a pattern before adding a ',
              'new one.' ) ;
      writeln ;
    end ;

7 : begin
      write( 'You presently have ', nwpn, ' weapon' ) ;
      if nwpn <> 1 then write( 's' ) ;
      writeln( ' defined.' ) ;
      writeln ;
    end ;

8 : begin
      write( 'You presently have ',npatt,' pattern' ) ;
      if npatt <> 1 then write( 's' ) ;
      writeln( ' defined.' ) ;
      writeln ;
    end ;
```

```
9 : begin
      writeln ;
      writeln ;
      writeln( 'D A T A   I N P U T   C O M P L E T E' ) ;
      writeln ;
      writeln ;
      writeln ;
      writeln( 'You are finished developing the  weapon/pattern  base.  The ',
               'next step is to put' ) ;
      writeln ;
      writeln( 'together a mission  using  AAPMSN  and  the two files you ',
               'have generated:   the' ) ;
      writeln ;
      writeln( 'target file  ( from AAPTGT )  and  the  weapon/pattern  file ',
               'from this program.' ) ;
      writeln ;
      writeln( 'AAPMSN  uses  these  files  in  combinations  you specify to ',
               'generate a single,' ) ;
      writeln ;
      writeln( 'user-named input file for the main program  --  AAPMOD.' ) ;
      pause ;
      writeln ;
      writeln ;
      writeln( 'W r i t i n g   t o   d i s k   . . .' ) ;
      writeln ;
    end ;

  end ;  { case }

end ;  { procedure info }

{================================================================================}
procedure wpnmatrix ;
  var i, j : integer ;
      msg : string[22] ;

  procedure writeheader ;   { header for weapons matrix }
  begin
    writeln ;
    write( 'Weapon #   !   j =  1' ) ;
    for j := 2 to nwpn do write( '       ', j ) ;
    writeln ;
    writeln( '----------!------------------------------------------',
             '------------' ) ;
  end ; { procedure writeheader }
```

```
begin { wpnmatrix }
  info( 2 ) ;
  if nhard > 8
    then msg := '   PRESS ANY KEY      '
    else msg := '                     ' ;
  writeheader ;
  writeln( 'Hardness   !   Radius of Circular Crater' ) ;
  writeln( '  Code     !' ) ;
  for i := 1 to nhard do begin
    write( msg[ ( 2 * i ) - 1 ],'  ', i:2, '  k=1 !' ) ;
    for j := 1 to nwpn do write( 2.0 / sqrt( pi ) * crtab[i,j,1]:9:1 ) ;
    writeln ;
    write( msg[ 2 * i ],'      k=2 !' ) ;
    for j := 1 to nwpn do write( 2.0 / sqrt( pi ) * crtab[i,j,2]:9:1 ) ;
    writeln ;
    if i = 9 then repeat until keypressed ;
  end ;
  writeln( '-----------!----------------------------------------',
           '-----------------' ) ;
  if i > 8 then writeheader ;
  pause ;
end ;  {  procedure wpnmatrix }

{=============================================================================}
procedure pattmatrix ;
  var i, j : integer ;
begin
  clrscr ;
  if npatt =  0 then begin
    writeln ;
    writeln( 'The attack pattern matrix is empty.' ) ;
    pause ;
  end ;
  for i := 1 to npatt do begin
    writeln ;
    write( 'Pattern #', i, ' has ', ipat[i,1],' ' ) ;

    case ipat[i,4] of

      0 : begin
            write( 'general purpose bomb' ) ;
            if ipat[i,1] <> 1 then write( 's' ) ;
          end ;

      1 : write( 'CBU ( rectangular footprint ) with ', ipat[i,2],
                 ' sub-munitions each' ) ;

      2 : write( 'CBU ( elliptical footprint ) with ', ipat[i,2],
                 ' sub-munitions each' ) ;
```

```
  3 : begin
        write( 'guided bomb' ) ;
        if ipat[i,1] <> 1 then write( 's' ) ;
      end ;

end ;  { case }

writeln( '.' ) ;
writeln( 'Crater Table Index:  j = ', ipat[i,3], '.  NOTE:  Error ',
         'values shown below are REP and DEP.' ) ;
writeln ;
if ipat[i,4] > 2
  then write( '  Optimal          Optimal        Near-miss        ',
              'Near-Miss      ' )
  else write( ' Delivery        Delivery       Ballistic        ',
              'Ballistic      ' ) ;
if ipat[i,4] in [ 1, 2 ]
  then writeln( ' Cannister ' )
  else writeln( '  Weapon' ) ;
writeln( 'Range Error   Deflection Error   Range Error    ',
         'Deflection Error   Reliability' ) ;
writeln( '-----------   ----------------   -----------    ',
         '----------------   -----------' ) ;
write( 0.675 * patt[i,1]:7:0 ) ;
for j := 2 to 4 do write( 0.675 * patt[i,j]:17:0 ) ;
writeln( patt[i,9]:17:3 ) ;
writeln ;
if ipat[i,4] in [ 1, 2 ] then begin
  writeln( '  Footprint        Footprint         Void            ',
           'Void          Bomblet' ) ;
  writeln( '   Length            Width           Length          ',
           'Width        Reliability' ) ;
  writeln( '  ---------        ---------         ------          ',
           '-----        -----------' ) ;
  write( 2 * patt[i,5]:8:0, 2 * patt[i,6]:18:0 ) ;
  if ipat[i,4] = 1
    then write( '             N/A            N/A ' )
    else write( 2 * patt[i,7]:16:0, 2 * patt[i,8]:16:0 ) ;
  writeln( patt[i,10]:17:3 ) ;
end ;
if ipat[i,4] = 3 then begin
  writeln( 'Gross Error     Gross Error       Probability of     ',
           'Cumulative Probability of' ) ;
  writeln( 'Range Error   Deflection Error    Optimal Guidance      ',
           'Near-Miss or better' ) ;
  writeln( '-----------   ----------------    ----------------     ',
           '-------------------------' ) ;
  writeln( 0.675 * patt[i,5]:8:0, 0.675 * patt[i,6]:15:0,
           patt[i,7]:22:3, patt[i,8]:24:3 ) ;
end ;
```

```
    if ipat[i,1] > 1 then begin
      writeln ;
      writeln ;
      writeln( 'Stick Impact Point Data:  ( Distance from mean impact ',
              'point. )' ) ;
      writeln ;
      write( ' Weapon #  1' ) ;
      for j := 2 to ipat[i,1] do write( j:6 ) ;
      writeln ;
      write( '        -----' ) ;
      for j := 2 to ipat[i,1] do write( ' -----' ) ;
      writeln ;
      write( '  Range' ) ;
      for j := 1 to ipat[i,1] do write( patt[ i, 2 * j + 9 ]:6:0 ) ;
      writeln;
      write( ' Offset' ) ;
      for j := 1 to ipat[i,1] do write( patt[ i, 2 * j + 10 ]:6:0 ) ;
      writeln ;
    end ;
    pause ;
  end ;
end ;  { procedure pattmatrix }

{================================================================}
procedure entwpn( i, j : integer ) ;
{ This procedure enters a weapon into the weapon matrix. }
begin
  crtab[i,j,1] := -1 ;
  crtab[i,j,2] := -1 ;
  writeln ;
  if nsqr = 0
    then writeln( 'Square craters: use 1/2 the length of a side.' )
    else writeln( 'Round craters:  use radius of a circular crater.' ) ;
  repeat
    writeln ;
    if i <= nhardp
      then write( 'Pavement code ',i,', weapon type ',j,':  Deny-TOL size ',
                  '==)  ' )
      else write( 'Building code ',i,', weapon type ',j,':  Near-miss size ',
                  '==)  ' ) ;
    realread( crtab[i,j,1] ) ;
  until crtab[i,j,1] >= 0 ;
  repeat
    writeln ;
    if i <= nhardp
      then write( 'Pavement code ',i,', weapon type ',j,': Deny-taxi size ',
                  '==)  ' )
      else write( 'Building code ',i,', weapon type ',j,':  Direct hit size ',
                  '==)  ' ) ;
    realread( crtab[i,j,2] ) ;
  until crtab[i,j,2] >= 0 ;
  clrscr ;
```

```
  if nsqr = 1 then begin
    crtab[i,j,1] := crtab[i,j,1] * sqrt( pi ) / 2 ;
    crtab[i,j,2] := crtab[i,j,2] * sqrt( pi ) / 2 ;
  end ;
  writeln ;
end ;  { procedure entwpn }

{=========================================================================}
procedure newwpn ;
{ This procedure creates a new weapons base to be written to a user defined
file name. }
  var badname : boolean ;
      i, ichc, j : integer ;
begin
  repeat
    processfile( fname, 'New Weapons' ) ;
    assign( diskfile, fname ) ;
    {$I-} reset( diskfile ) ;  {$I+ check to see if diskfile exists }
    badname := ioresult = 0 ;     { the "new" file already exists }
    if badname then begin
      write( 'File (', fname, ') already exists.  Erase?' ) ;
      query( ichc, 1 ) ;
      if ichc = 0 then begin
        write( 'Are you SURE you want to erase file (',fname, ') ?' ) ;
        query( ichc, 1 ) ;
        if ichc = 0 then begin
          rewrite( diskfile ) ;
          badname := false ;
        end ;
      end ;
    end ;
  until not badname ;
  info( 3 ) ;
  repeat
    clrscr ;
    nhard := -1 ;
    nhardp := -1 ;
    repeat
      writeln ;
      writeln ;
      writeln( 'Enter the total number of hardness levels you specified' ) ;
      write( 'in program AAPTGT ==>  ' ) ;
      intread( nhard ) ;
    until chkgood( nhard, 11, 1 ) ;
    repeat
      writeln( 'Enter the number of hardness levels reserved for pavements' ) ;
      write( 'in program AAPTGT ==>  ' ) ;
      intread( nhardp ) ;
      clrscr ;
    until chkgood( nhardp, nhard, 0 ) ;
    hardness ;
    write( 'Are these correct?' ) ;
    query( ichc, 1 ) ;
```

```
  until ichc = 0 ;
  repeat
    write( 'Enter number of different types of warheads ( max is 6 ) ==)  ' ) ;
    intread( nwpn ) ;
  until chkgood( nwpn, 6, 1 ) ;
  info( 4 ) ;
  query( nsqr, 8 ) ;
  for i := 1 to nhard do
    for j := 1 to nwpn do begin
      entwpn( i, j ) ;
    end ;
  writeln ;
end ;  { procedure newwpn }

{==============================================================================}
procedure delwpn ;
{ This procedure deletes a weapon type from the weapon matrix. }
  var i, ichc, j, jchc, k, pattern : integer ;
      pattok : boolean ;
      delthis : array [1..12] of boolean ;
begin
  pattok := true ;
  for i := 1 to 12 do delthis[i] := false ;
  if nwpn = 0
    then writeln( 'There are no more weapons left to delete.' )
    else begin
        repeat
          write( 'Enter weapon number to delete ==)  ' ) ;
          intread( ichc ) ;
        until chkgood( ichc, nwpn, 1 ) ;
        clrscr ;
        writeln ;
        for i := 1 to npatt do if ipat[i,3] = ichc then begin
          write( i:4 ) ;     { pattern references the weapon to be deleted }
          delthis[i] := true ;   { flag pattern to be deleted }
          pattok := false ;
        end ;
        if pattok
          then begin
              writeln( 'There are no patterns defined for weapon #', ichc:2,
                       ', so no patterns will be deleted' ) ;
              writeln( 'from the pattern matrix.  However, applicable crater ',
                       'table index values will' ) ;
              writeln( 'be adjusted as necessary.' ) ;
            end
          else begin
              writeln( '   <== W A R N I N G' ) ;
              writeln ;
              writeln( 'The above listed patterns are defined  for the weapon ',
                       'you wish to delete.' ) ;
              writeln( 'Therefore, if you delete weapon #', ichc:2, ', these ',
                       'patterns will be deleted also.' ) ;
```

```
            writeln( 'Furthermore,  applicable crater table index values  ',
                     'will  be  adjusted as' ) ;
            writeln( 'necessary.' ) ;
          end ;
      writeln ;
      writeln( 'Would you like to review the pattern matrix before' ) ;
      write( 'deleting weapon # ', ichc, '?' ) ;
      query( jchc, 1 ) ;
      if jchc = 0 then pattmatrix ;
      writeln ;
      write( 'Are you SURE you want to delete weapon # ',ichc,'?' ) ;
      query( jchc, 1 ) ;
      clrscr ;
      writeln ;
      if jchc = 0
        then begin
            for pattern := 12 downto 1 do if delthis[pattern] then begin
              npatt := npatt - 1 ;
              if pattern < npatt + 1 then
                for i := pattern to npatt do begin
                  for j := 1 to 4 do ipat[i,j] := ipat[i + 1, j] ;
                  for j := 1 to 34 do patt[i,j] := patt[i + 1, j] ;
                end ;
            end ;
            for i := 1 to npatt do if ipat[i,3] > ichc
              then ipat[i,3] := ipat[i,3] - 1 ; { pattern crtab index info }
            nwpn := nwpn - 1 ;
            if ichc < nwpn + 1 then
              for i := 1 to nhard do
                for j := ichc to nwpn do
                  for k := 1 to 2 do crtab[i,j,k] := crtab[i,j+1,k] ;
            writeln( 'Weapon # ', ichc, ' deleted.' ) ;
          end
        else writeln( 'Weapon # ', ichc, ' NOT deleted.' ) ;
    end ;
  writeln ;
end ;  { procedure delwpn }

{==========================================================================}
procedure addwpn ;
{ This procedure adds a weapon type to the weapons matrix. }
  var i : integer ;
begin
  if nwpn >= 6
    then info( 5 )
    else begin
        nwpn := nwpn + 1 ;
        query( asqr, 8 ) ;
        for i := 1 to nhard do entwpn( i, nwpn ) ;
    end ;
end ;  { procedure addwpn }
```

```
{==============================================================}
procedure inswpn ;
{ This procedure inserts a new weapon into the crater table. }
  var i, j, k, wpnpos : integer ;
begin
  clrscr ;
  writeln ;
  case nwpn of

    0 : addwpn ;

 1..5 : begin
          writeln( 'The  new  weapon  will  be  inserted  at the position ',
                   'you specify,  displacing' ) ;
          writeln( 'previous entries.  The pattern matrix will be adjusted ',
                   'to reflect the modified' ) ;
          writeln( 'weapon numbers ( Crater Table Index j ).' ) ;
          writeln ;
          info( 7 ) ;
          repeat
            write( 'New weapon to be inserted at position #  ==>  ' ) ;
            intread( wpnpos ) ;
            writeln ;
          until chkgood( wpnpos, nwpn + 1, 1 ) ;
          if wpnpos = nwpn + 1
            then addwpn
            else begin
                query( nsqr, 8 ) ;
                for j := nwpn downto wpnpos do { make room for insert }
                  for i := 1 to nhard do
                    for k := 1 to 2 do crtab[ i, j+1, k ] := crtab[ i,j,k ] ;
                nwpn := nwpn + 1 ;
                for i := 1 to nhard do entwpn( i, wpnpos ) ;  { enter new wpn }
                for i := 1 to npatt do  { adjust pattern matrix crtab index j }
                  if ipat[i,3] >= wpnpos then ipat[i,3] := ipat[i,3] + 1 ;
              end ;
        end ;

    6 : info( 5 )

  end ; { case }

end ;  {  procedure inswpn  }
```

```
{==========================================================================}
procedure modwpn ;
{ This procedure redefines an existing weapon without altering the
corresponding pattern in the pattern matrix.  }
  var i, modpos : integer ;
begin
  if nwpn = 0
    then begin
        writeln( 'There are no weapons in the matrix to redefine.  You now ',
                 'must add a weapon.' ) ;
        writeln ;
      end
    else begin
        writeln( 'Now select the weapon you wish to redefine.  Note that ',
                 'any corresponding' ) ;
        writeln( 'patterns in the attack pattern matrix will remain ',
                 'unchanged.' ) ;
        writeln ;
        info( 7 ) ;
        repeat
          write( 'Weapon to be redefined is # ==>  ' ) ;
          intread( modpos ) ;
          writeln ;
        until chkgood( modpos, nwpn, 1 ) ;
        query( nsqr, 8 ) ;
        for i := 1 to nhard do entwpn( i, modpos ) ;  { enter weapon info }
      end ;
end ; { procedure modwpn }

{==========================================================================}
procedure oldwpn ;
{ This procedure updates an existing weapons base. }
  var i, ichc, j : integer ;
begin
  repeat processfile( fname, 'Old Weapons' ) until filexists( fname ) ;
  writeln ;
  writeln( '        R e a d i n g   f i l e   ( ', fname, ' ) . . .' ) ;
  writeln ;
  writeln ;
  read( diskfile, nhard, nhardp, nwpn ) ;
  for i :=  1 to nhard do begin
    for j := 1 to nwpn do read( diskfile, crtab[i,j,1] ) ;
    for j := 1 to nwpn do read( diskfile, crtab[i,j,2] ) ;
  end ;
  read( diskfile, npatt ) ;
  for i := 1 to npatt do begin
    for j := 1 to 4 do read( diskfile, ipat[i,j] ) ;
    for j := 1 to 10 do read( diskfile, patt[i,j] ) ;
    for j := 11 to ( 2 * ipat[i,1] + 10 ) do read( diskfile, patt[i,j] ) ;
  end ;
  close( diskfile ) ;
  info( 7 ) ;
  hardness ;
```

```
  writeln ;
  writeln( 'The only way to update the number of hardness levels is by ',
           'updating the ' ) ;
  writeln( 'target base using AAPTGT.' ) ;
  pause ;
  writeln ;
  writeln( 'This section adds and deletes weapons to and from the ',
           'weapons base.' ) ;
  writeln ;
  repeat
    query( ichc, 2 ) ;
    case ichc of
      0 : delwpn ;
      1 : addwpn ;
      2 : inswpn ;
      3 : modwpn ;
      4 : wpnmatrix ;
      5 : pattmatrix ;
    end ; { case }
  until ichc = 6 ;
end ;  { procedure oldwpn }

{==========================================================================}
procedure patdat( i : integer ) ;
{ This procedure collects preliminary pattern data. }
begin
  ipat[i,1] := 1 ; { default number of weapons in the pattern }
  ipat[i,2] := 1 ; { default number of submunitions }
  clrscr ;
  writeln ;
  writeln( 'Describe weapons delivery pattern number ', i, '.  Enter:' ) ;
  writeln ;
  query( ipat[i,4], 4 ) ;
  if ipat[i,4] < 3
    then repeat
      write( 'Number of weapons in the pattern ( maximum 12 ) ==>  ' ) ;
      intread( ipat[i,1] ) ;
    until chkgood( ipat[i,1], 12, 1 ) ;
  repeat
    if ipat[i,4] in [ 1, 2 ] then write( 'Cannister' ) else write( 'Weapon' ) ;
    write( ' fuze reliability ==>  ' ) ;
    realread( patt[i,9] ) ;
  until probchk( patt[i,9] ) ;
  patt[i,10] := patt[i,9] ;
  writeln ;
  repeat
    write( 'Enter the weapon''s Crater Table Index  ==>  ' ) ;
    intread( ipat[i,3] ) ;
  until chkgood( ipat[i,3], mwpn, 1 ) ;
end ;  { procedure patdat }
```

```
{==========================================================================}
procedure delpat ;
{ This procedure deletes an attack pattern from the pattern matrix. }
  var i, ichc, j, jchc : integer ;
begin
  if npatt = 0
    then writeln( 'There are no patterns left to delete.' )
    else begin
        repeat
          write( 'Enter pattern number to delete ==>  ' ) ;
          intread( ichc ) ;
        until chkgood( ichc, npatt, 1 ) ;
        write( 'Are you SURE you want to delete pattern ',ichc,'?' ) ;
        query( jchc, 1 ) ;
        clrscr ;
        writeln ;
        if jchc = 0
          then begin
              npatt := npatt - 1 ;
              if ichc < npatt + 1 then
                for i := ichc to npatt do begin
                  for j := 1 to 4 do ipat[i,j] := ipat[i + 1, j] ;
                  for j := 1 to 34 do patt[i,j] := patt[i + 1, j] ;
                end ;
              writeln( 'Pattern ', ichc, ' deleted.' ) ;
            end
          else writeln( 'Pattern ', ichc, ' NOT deleted.' ) ;
      end ;
  writeln ;
end ;  { procedure delpat }

{==========================================================================}
procedure guided( i : integer ) ;
{ This procedure enters information for guided munitions. }
  var ichc, j : integer ;
begin
  query( ichc, 7 ) ;
  writeln ;
  writeln( 'Enter:'  ) ;
  if ichc = 1
    then begin
        write( 'Optimal guidance REP ==>  ' ) ;
        realread( patt[i,1] ) ;
        write( 'Optimal guidance DEP ==>  ' ) ;
        realread( patt[i,2] ) ;
        write( 'Near-miss REP ==>  ' ) ;
        realread( patt[i,3] ) ;
        write( 'Near-miss DEP ==>  ' ) ;
        realread( patt[i,4] ) ;
        write( 'Gross error REP ==>  ' ) ;
        realread( patt[i,5] ) ;
        write( 'Gross error DEP ==>  ' ) ;
        realread( patt[i,6] ) ;
```

```
      for j := 1 to 6 do patt[i,j] := patt[i,j] / 0.675 ;
     end
   else begin
      write( 'Optimal guidance, range error sigma ==> ' ) ;
      realread(  patt[i,1] ) ;
      write( 'Optimal guidance, deflection error sigma ==>  ' ) ;
      realread( patt[i,2] ) ;
      write( 'Near-miss, range error sigma ==>  ' ) ;
      realread( patt[i,3] ) ;
      write( 'Near-miss, deflection error sigma ==>  ' ) ;
      realread( p..t[i,4] ) ;
      write( 'Gross error, range error sigma ==>  ' ) ;
      realread( patt[i,5] ) ;
      write( 'Gross error, deflection error sigma ==>  ' ) ;
      realread( patt[i,6] ) ;
     end ;
  clrscr ;
  writeln ;
  repeat
    write( 'Probability of optimal guidance ==>  ' ) ;
    realread( patt[i,7] ) ;
  until probchk( patt[i,7] ) ;
  repeat
    write( 'Cumulative probability of near-miss guidance or better ==>  ' ) ;
    realread( patt[i,8] )  ;
    if ( patt[i,8] < patt[i,7] ) or ( patt[i,8] > 1 ) then begin
      writeln( 'Cumulative probability must be equal to or greater than ',
               'the probability' ) ;
      writeln( 'of optimal guidance entered above.' ) ;
      writeln ;
      writeln( 'Enter a value between ', patt[i,7]:7:5, ' and 1.' ) ;
      writeln ;
    end ;
  until ( patt[i,8] >= patt[i,7] ) and ( patt[i,8] <= 1 ) ;
end ;  { procedure guided }

{=========================================================================}
procedure manstk( i : integer ) ;
{ This procedure manually enters a stick of weapons. }
  var jdef, jrng, lj : integer ;
begin
  clrscr ;
  writeln ;
  writeln( 'For each weapon, enter its signed (+/-) range and deflection ',
          'position' ) ;
  writeln( 'within the stick, referenced to the aimpoint.' ) ;
  writeln ;
  for lj := 1 to ipat[i,1] do begin
    jrng := 2 * lj + 9 ;
    jdef := jrng + 1 ;
    write( 'Weapon ', lj, ' range position ==>  ' ) ;
    realread( patt[i,jrng] ) ;
```

```
    write( 'Weapon ', lj, ' deflection position ==> ' ) ;
    realread( patt[i,jdef] ) ;
    writeln ;
  end ;
end ;  { procedure manstk }

{=====================================================================}
procedure atostk( i : integer ) ;
{ This procedure calculates impact points for a stick of weapons. }
  var j, jdef, jrng, lj, np : integer ;
      df, dt, fps, ft, sstart, stickl, tas, theta, wft : real ;
begin
  write( 'Enter true airspeed ( knots ) ==> ' ) ;
  realread( tas ) ;
  fps := 1.6875 * tas ;
  write( 'Enter dive angle in ( degrees ) ==> ' ) ;
  realread( theta ) ;
  theta := theta * pi / 180 ; { convert dive angle to radians }
  query( np, 5 ) ;  { np: returned as 1 for single release, 2 for pairs }
  np := round( 0.25 + ipat[i,1] / np ) ; { calculate # pulses based on # wpns }
  if np > 1
    then begin
        write( 'Enter intervalometer setting ( milliseconds ) ==> ' ) ;
        realread( dt ) ;
        dt := dt / 1000 ; { convert dt to seconds }
      end
    else dt := 0 ;
  write( 'Enter width of weapon stick (feet) ==> ' ) ;
  realread( ft ) ;
  wft := ft / 2 ;
  df := fps * dt * cos( theta ) ;
  stickl := df * ( np - 1 ) ;
  sstart := -0.5 * stickl - df ;
  if ipat[i,1] = np { dropping in singles }
    then for lj := 1 to ipat[i,1] do begin
        jrng := 2 * lj + 9 ;
        jdef := jrng + 1 ;
        patt[i,jrng] := sstart + lj * df ;
        patt[i,jdef] := wft ;
        if lj mod 2 = 0 then patt[i,jdef] := patt[i,jdef] * -1 ;
      end
    else begin { dropping in pairs }
        j := 2 ;
        while j <= ipat[i,1] do begin
          patt[i, 2 * j + 7] := sstart + df * j / 2 ;
          patt[i, 2 * j + 8] := wft ;
          patt[i, 2 * j + 9] := sstart + df * j / 2 ;
          patt[i, 2 * j + 10] := -1 * wft ;
          j := j + 2 ;
        end ;
      end ;
end ;  { procedure atostk }
```

```
{=================================================================}
procedure descbu( i : integer ) ;
{ This procedure enters descriptive information for CBU patterns. }
  var al, aw, vl, vw : real ;
begin
  clrscr ;
  writeln ;
  write( 'Enter the number of bomblets per CBU cannister  ==>  ' ) ;
  intread( ipat[i,2] ) ;
  repeat
    write( 'Bomblet fuze reliability ==>  ' ) ;
    realread( patt[i,10] ) ;
  until probchk( patt[i,10] ) ;
  writeln( 'Describe CBU bomblet distribution.  Enter:' ) ;
  writeln ;
  write( 'Dispenser ground coverage, full length ( range ) ==>  ' ) ;
  realread( al ) ;
  patt[i,5] := al / 2 ;
  write( 'Dispenser ground coverage, full width ( deflection ) ==> ' ) ;
  realread( aw ) ;
  patt[i,6] := aw / 2 ;
  if ipat[i,4] > 1
    then begin
        write( 'Void length ( range ) ==>  ' ) ;
        realread( vl ) ;
        patt[i,7] := vl / 2 ;
        write( 'Void width ( deflection ) ==>  ' ) ;
        realread( vw ) ;
        patt[i,8] := vw / 2 ;
      end
    else begin
        patt[i,7] := 0 ;
        patt[i,8] := 0 ;
      end ;
end ;  { procedure descbu }

{=================================================================}
procedure entpat( i : integer ) ;
{ This procedure enters a delivery pattern at index i in the pattern matrices.}
  var j, opatt : integer ;
begin
  patt[i,11] := 0 ;
  patt[i,12] := 0 ;
  patdat( i ) ;
  if ipat[i,4] >= 3
    then guided ( i )
```

```
    else begin
      clrscr ;
      writeln ;
      writeln( 'Describe error distributions with range and deflection ',
               'errors probable.' ) ;
      writeln ;
      writeln( 'Enter:' ) ;
      writeln ;
      write( 'Delivery range error probable ==> ' ) ;
      realread( patt[i,1] ) ;
      write( 'Delivery deflection error probable ==>  ' ) ;
      realread( patt[i,2] ) ;
      write( 'Ballistic range error probable ==>  ' ) ;
      realread( patt[i,3] ) ;
      write( 'Ballistic deflection error probable ==>  ' ) ;
      realread( patt[i,4] ) ;
      for j := 1 to 4 do patt[i,j] := patt[i,j] / 0.675 ;
      if ipat[i,1] > 1 then begin
        query( opatt, 6 ) ;
        if opatt = 0
          then manstk( i )
          else atostk( i ) ;
      end ;
      if ipat[i,4] > 0
        then descbu( i )
        else for j := 5 to 8 do patt[i,j] := 0 ;
    end ;
  clrscr ;
  writeln ;
end ;  { procedure entpat }

{=======================================================================}
procedure addpat ;
{ This procedure adds an attack pattern to the pattern matrix. }
begin
  if npatt >= 12
    then info( 6 )
    else begin
      npatt := npatt + 1 ;
      entpat( npatt ) ;
    end ;
end ;  { procedure addpat }
```

```
{==================================================================}
procedure newpat ;
{ This procedure adds newly defined pattern data to a new or existing weapons
base. }
  var i : integer ;
begin
  repeat
    writeln( 'Enter the number of different weapon delivery patterns ',
             'to be entered' ) ;
    write( '( Maximum = 12 )  ==>  ' ) ;
    intread( npatt ) ;
  until chkgood( npatt, 12, 1 ) ;
  for i := 1 to npatt do entpat( i ) ;
end ;  { procedure newpat }

{==================================================================}
procedure inspat ;
{ This procedure inserts a new pattern into the pattern matrix at position
patpos. }
  var i, j, patpos : integer ;
begin
  clrscr ;
  writeln ;
  case npatt of

    0 : addpat ;

1..11 : begin
        writeln( 'The new pattern will be inserted at the position you ',
                 'specify, displacing ' ) ;
        writeln( 'previous entries.' ) ;
        writeln ;
        info( 8 ) ;
        repeat
          write( 'New pattern to be inserted at position # ==>  ' ) ;
          intread( patpos ) ;
          writeln ;
        until chkgood( patpos, npatt + 1, 1 ) ;
        if patpos = npatt + 1
          then addpat
          else begin
              for i := npatt downto patpos do begin { make room for insert }
              for j := 1 to 4 do ipat[i+ 1,j]  := ipat[i,j] ;
              for j := 1 to 34 do patt[i + 1,j] := patt[i,j] ;
            end ;
            npatt := npatt + 1 ;
            entpat( patpos ) ;
          end ;
      end ;

   12 : info( 6 ) ;
  end ; { case }
end ;  {  procedure inswpn  }
```

```
{==================================================================}
procedure oldpat ;
{ This procedure modifies the existing pattern base after the existing weapons
base has been modified. }
  var i, ichc, j, p : integer ;
begin
  writeln( 'This section adds and deletes attack patterns to/from the ',
           'pattern matrix.' ) ;
  writeln ;
  repeat
    query( ichc, 3 ) ;
    case ichc of
      0 : delpat ;
      1 : addpat ;
      2 : inspat ;
      3 : wpnmatrix ;
      4 : pattmatrix ;
    end ;  { case }
  until ichc = 5 ;
  backup := fname ;                          { create a filename equal to  }
  p := pos( '.',backup ) ;                   { specified filename except   }
  if p <> 0 then delete( backup, p, 4 ) ;    { with a .BAK filetype.       }
  backup := backup + '.BAK' ;                {                             }
  assign( diskfile, backup ) ;
  {$I-} reset( diskfile ) ; {$I+}            { see if old .BAK file exists }
  if ioresult = 0 then erase ( diskfile ) ;  { if yes, erase old .BAK file }
  assign ( diskfile, fname ) ;               { fname = original file       }
  rename( diskfile, backup ) ;       { original file now saved with .BAK name }
  clrscr ;
  writeln ;
  writeln ;
  writeln( 'The original version of < ', fname, ' > has been saved in' ) ;
  writeln( 'a file named < ', backup, ' >' ) ;
  writeln ;
end ;  { procedure oldpat }
```

```
{=====================================================================}
procedure outdat ;
{ This procedure writes the weapons base to a new file with the name specified
by the user at program invocation.  If  the file specified was an existing file
named <filename>.<filetype>, the old version was saved with the name
<filename.BAK> by procedure oldpat.                                    }
  var i, j, jr : integer ;
begin
  info( 9 ) ;
  assign( diskfile, fname ) ;       { reassign new file to write as the     }
  rewrite( diskfile ) ;             { original, user specified filename     }
  writeln( diskfile, nhard:5, nhardp:5, nwpn:5 ) ;
  for i := 1 to nhard do begin
    for j := 1 to nwpn do write( diskfile, crtab[i,j,1]:9:1 ) ;
    writeln( diskfile ) ;
    for j := 1 to nwpn do write( diskfile, crtab[i,j,2]:9:1 ) ;
    writeln( diskfile ) ;
  end ;
  writeln( diskfile, npatt:5 ) ;
  for i := 1 to npatt do begin
    for j := 1 to 4 do write( diskfile, ipat[i,j]:5 ) ;
    writeln( diskfile ) ;
    for j := 1 to 8 do write( diskfile, patt[i,j]:8:1 ) ;
    write( diskfile, patt[i,9]:7:3, patt[i,10]:7:3 ) ;
    writeln( diskfile ) ;
    for j := 1 to ipat[i,1] do begin
      jr := 2 * j + 9 ;
      writeln( diskfile, patt[i,jr]:8:1, patt[i, jr + 1]:8:1 ) ;
    end ;
  end ;
  close( diskfile ) ;
  writeln ;
  writeln( 'The new file created is named < ', fname, ' >' ) ;
end ;  { procedure outdat }

{=================  M A I N   P R O G R A M  =================}
begin
init ;
info( 1 ) ;
if ichc = 0
  then begin
      newwpn ;
      newpat ;
    end
  else begin
      oldwpn ;
      oldpat ;
    end ;
outdat ;
writeln ;
writeln( 'E N D   O F   P R O G R A M' ) ;
writeln ;
end.
```

```
{========================================================================}
{==================== file AAPMSN.PAS    10 Feb 85 =======================}
{========================================================================}

{$IAAPMSN1.PAS  compiler directive to include file AAPMSN1.PAS}
```

```
{==========================================================================}
{===================  FILE AAPMSN1.PAS    10 Feb 85  ======================}
{==========================================================================}
program aapmsn ;

type str14 = string[14] ;

var diskfile : text ;

    fname1, fname2, fname3, seed : str14 ;

    crit : array[1..112,1..2] of real ;
    crtab : array[1..11,1..6,1..2] of real ;
    pass : array[1..32,1..5] of real ;
    patt : array[1..12,1..34] of real ;
    tgt : array[1..112,1..5] of real ;

    acft : array[1..64,1..2] of integer ;
    ipass, opt : array[1..32,1..2] of integer ;
    ipat : array[1..12,1..4] of integer ;
    itgt : array[1..112,1..3] of integer ;
    npx : array[1..32] of integer ;

    irepr, iseed, kac, lv, mxptch, nacft, narea, nbldg, ncp, nelt, nflag3,
    nhard, nhardp, npass, npatt, npprcw, nsamp, nsampt, nsqr, ntgps, nwpn
      : integer ;

    error, zalpha : real ;

    passflag : boolean ;

{    variable key:
      crit[i,j] - array with critical distances (ft)
                  i: target element number (112 max)
                  j: 1 - min clear takeoff length (0.0 => taxi only)
                     2 - min clear takeoff(taxi) width
      crtab[i,j,k] - crater tabulated data (ft)
                  i: surface type (hardness code) (11 max)
                  j: weapons type (6 max)
                  k: crater radii subdivided by surface type (ft)
                     1 - if pave: tol crater size; bldg: near miss size
                     2 - if pave: taxiway crater size; bldg: direct
                                    hit size
      error - level of statistical significance for analysis
      fname - character variable for file names
      hi - high value for integer range comparison
      i,j - loop counters
      ichc - choice flag
      ipass[i,j] - supplementary mission data
                  i: pass number (32 max)
                  j: 1 - pattern number flown this pass
                    2 - next pass (absolute) this A/C will fly
```

ipat[i,j] - supplementary pattern data
i: pattern number (12 max)
j: 1 - number of weapons delivered (12 max)
2 - number of submunitions per weapon
3 - weapon code (crtab index number)
4 - pattern shape
0 = general purpose bomb
1 = cbu (rectangular)
2 = cbu (elliptical)
3 = guided bomb
irepr - code for repair priorities
0 = all tol strip in order of target number
1 = easiest tol strip first, rest in order
2 = repair only the easiest tol strip
10 = all pavements in order of target number
11 = all approaches and easiest tol strip first, followed by others in target order
12 = all approaches and only the easiest tol strip
iseed - random number input number
itgt[i,j] - array with supplementary target data
i: target element number (112 max)
j: 1 - 1=pavement, 0=building
2 - hardness code for target (11 max)
3 - belongs to target group number
ix - dummy argument for procedure call option
jchc - choice flag
jd,jr - intermediate variables (deflection and range)
jdef - deflection counter (index in pattern matrix)
jrng - range counter (index in pattern matrix)
kac - aircraft number performing current pass
lo - lower value of integer range comparison
lv - number of taxi surfaces (30-ncp max)
m - intermediate storage for nhard
mxptch - number of runway patches available
n - intermediate storage for nhardp
nac - number of aircraft in mission
narea - flag: 0 = compute tol damage, 1 = don't compute
nbldg - number of buildings (112-ncp-lv max)
ncp - number of takeoff/landing capable surfaces (3 max)
nelt - total number of targets (112 max)
nflag3 - 0 = no effect, 1 = optimize monte carlo iterations (if>200)
nhard - total hardness codes for buildings plus surfaces (11 max)
nhardp - hardness codes for surfaces (nhard max)
npass - number of attacks/passes on target (32 max)
npatt - number of patterns (12 max)
nopprcw - min taxi width required, 0.0 suppresses search for taxiways
npx[i] - target group of target being attacked this pass
i: pass number (32 max)
nsamp - number of monte carlo iterations
nsampt - number of monte carlo iterations between intermediate reports
nsqr - crater input flag (0=> square, 1=> round)
ntgps - number of distinct target groups
nval - dummy argument for integer range comparisons

```
        nvals - intermediate variable
        nwpn - number of different weapon types (6 max)
        opt[i,j] - optional second pass data
                   i: pass number (32 max)
                   j: 1 - 0 = first pass for this A/C
                          1 = second pass for this A/C
                      2 - A/C number flying this pass
        pass[i,j] - mission data
                   i: pass number (32 max)
                   j: 1 - aimpoint x-coord (ft)
                      2 - aimpoint y-coord (ft)
                      3 - axis of attack (deg)
                      4 - probability A/C reaches target
                      5 - probability A/C survives to reattack
        patt[i,j] - pattern data
                   i: pattern number (12 max)
                   j: 1 - aiming range error (ft)
                      2 - aiming deflection error (ft)
                      3 - ballistic range error (ft)
                      4 - ballistic deflection error (ft)
                      5 - cbu half pattern length (ft)
                      6 - cbu half pattern width (ft)
                      7 - cbu half void length (ft)
                      8 - cbu half void width (ft)
                      9 - fuze reliability
                     10 - bomblet reliability
        rat - average number of attacks per airplane (2.0 max)
        tgt[i,j] - primary target data
                  i:  target element number (112 max)
                  j:  1 - x-coord (ft)
                      2 - y-coord (ft)
                      3 - main axis (deg)
                      4 - length (ft)
                      5 - width (ft)
        wait - temporary storage variable
        zalpha - standard normal statistic                              }

{=================================================================}
procedure realread( var r : real ) ;
{ procedure to read a real value with error checking. }
begin
  repeat {$I-} readln( r ) {$I+} until ( ioresult = 0 ) ;
  writeln ;
end ;  { procedure realread }

{=================================================================}
procedure intread( var i : integer ) ;
{ procedure to read an integer value with error checking. }
begin
  repeat {$I-} readln( i ) {$I+} until ( ioresult = 0 ) ;
  writeln ;
end ;  { procedure intread }
```

```
{==========================================================================}
procedure pause ;
{ delays crt output until a key is pressed }
begin
  writeln ;
  write( '          p r e s s   a n y   k e y   t o   c o n t i n u e  ' ) ;
  repeat until Keypressed ;
  clrscr ;
end ;  { procedure pause }

{==========================================================================}
function chkgood( nval, hi, lo : integer ) : boolean ;
{ this function returns false when nval is out of range }
begin
  if ( nval < lo ) or ( nval > hi)
    then begin
        writeln ;
        writeln( 'Value must be between ',lo,' and ',hi ) ;
        writeln ;
        chkgood := false ;
      end
    else chkgood := true ;
end ; { function chkgood }

{==========================================================================}
function probchk( prob : real ) : boolean ;
{ this function returns false when prob is out of range ( 0 to 1 ) }
begin
  if ( prob < 0 ) or ( prob > 1 )
    then begin
        writeln ;
        writeln( 'Enter a value between 0 and 1.' ) ;
        writeln ;
        probchk := false ;
      end
    else probchk := true ;
end ; { function probchk }

{==========================================================================}
procedure query( var ichc: integer ) ;
{ this procedure returns response to a yes-no question
  ichc set to 0 for yes, and 1 for no or ( return ).    }
  var ans : char ;
begin
  write( '  (y/n)  ==>  ' ) ;
  repeat readln( ans ) until ( upcase( ans ) in [ 'Y','N' ] ) or eoln ;
  writeln ;
  if eoln or ( upcase( ans ) = 'N' )
    then ichc := 1
    else ichc := 0 ;
end ;  { procedure query }
```

```
{=====================================================================}
procedure processfile( var fname : str14; which : str14 ) ;
{ initial processing of file name entered by user }
  var dot, goodname : boolean ;
      p, size, dotpos : integer ;
begin
  clrscr ;
  writeln ;
  writeln( 'For the following entries, allowable filenames may contain only ',
          'capital letters' ) ;
  writeln( 'and numbers.   If you enter lower case letters, they will be ',
          'treated as if they' ) ;
  writeln( 'are upper case.  The format must be as follows:' ) ;
  writeln ;
  writeln( '          D:XXXXXXXX.XXX' ) ;
  writeln ;
  writeln( '   where   D: is an optional drive specifier,' ) ;
  writeln( '          X is a digit ( 0 - 9 ) or a letter ( A - Z ),' ) ;
  writeln( '          XXXXXXXX is a filename 1 to 8 characters long, and' ) ;
  writeln( '          .XXT is an optional 1 to 3 character file type',
          ' ( not .BAK ).' ) ;
  writeln ;
  writeln( 'Examples of valid filenames:  MSN    MSN.1    AAPMOD.DTA    ',
          'B:1JAN84.AAP' ) ;
  writeln ;
  writeln ;
  writeln( 'If an existing file you need to use contains  characters  not ',
          'compatible with' ) ;
  writeln( 'the above format,  you will need to ABORT this program by ',
          'pressing control-C,' ) ;
  writeln( 'and then rename the file to make it compatible.' ) ;
  pause ;
  repeat
    goodname := true ;
    dot := false ;
    repeat
      writeln ;
      write( 'Enter name for the ', which, ' File ==>  ' ) ;
      readln( fname ) ;
      writeln ;
      for p := 1 to 14 do fname[p] := upcase( fname[p] ) ;
      repeat                                  { delete blanks from }
        p := pos( ' ', fname ) ;              { filename           }
        if p <> 0 then delete( fname, p, 1 ) ;
      until p = 0 ;
    until fname <> '' ;              { trap for carriage return/blank filename}
    dotpos := pos( '.', fname ) ;    { filetype can only be 3 characters long }
    if dotpos <> 0 then delete( fname, dotpos + 4, 14 ) ;
    size := length( fname ) ;
    p := pos( ':', fname ) ;
```

```
if ( p = 2 )
  then begin

      if ( not ( dotpos in [ 0, 4..11 ] ) )
                         or
         ( not ( fname[1] in [ 'A'..'Z' ] ) )
                         or
                 ( size < 3 )
                         or
         ( ( dotpos = 0 ) and ( size > 10 ) )
                        then
               goodname := false ;

      if goodname then for p := 3 to size do begin
        if not ( fname[p] in [ '0'..'9', 'A'..'Z', '.' ] )
          then goodname := false ;
        if dot and ( fname[p] = '.' ) then goodname := false ;
        if fname[p] = '.' then dot := true ;
      end ;
    end
  else begin

      if ( not ( dotpos in [ 0, 2..9 ] ) )
                         or
                 ( size < 0 )
                         or
         ( ( dotpos = 0 ) and ( size > 8 ) )
                        then
               goodname := false ;

      if goodname then for p := 1 to length( fname ) do begin
        if not ( fname[p] in [ '0'..'9', 'A'..'Z', '.' ] )
          then goodname := false ;
        if dot and ( fname[p] = '.' ) then goodname := false ;
        if fname[p] = '.' then dot := true ;
      end ;
    end ;
if pos( '.BAK', fname ) <> 0 then begin
  goodname := false ;
  writeln( 'You may not enter a filename with the filetype ".BAK" .' ) ;
  writeln ;
  writeln( 'To use the backup .BAK file, rename it before executing ',
          'this program.' ) ;
  writeln ;
end ;

writeln ;
if goodname
  then begin
      write( 'Confirm the correct filename is ( ', fname, ' ) ?' ) ;
      query( p ) ;
      if p = 1 then goodname := false ;
    end
```

```
      else begin
         writeln( 'Invalid filename < ',fname, ' >' ) ;
         writeln ;
         writeln( 'Please try again, or press control-C to ABORT this ',
                  'program.' ) ;
       end ;
  until goodname ;
end ;  { procedure processfile }

{===========================================================================}
function filexists( fname : str14 ) : boolean ;
{ Checks to see if file fname exists on disk. Returns true if it does. }
  var exists : boolean ;
begin
  assign( diskfile, fname ) ;
  {$I-} reset( diskfile ) ; {$I+  this checks if the file exists}
  exists := ioresult = 0 ;
  if not exists then begin
    writeln( 'File < ', fname, ' > does not exist.  Be sure to specify the ',
             'correct drive ' ) ;
    writeln( 'and the correct file name.' ) ;
    writeln ;
    writeln( '                To ABORT this program, press control-C.' ) ;
    pause ;
  end ;
  filexists := exists ;
end ;  { function filexists }

{===========================================================================}
procedure info( ix : integer ) ;
{ This procedure declutters the main program by placing some text in a
  central location. }
begin
  case ix of

  1 : begin
       clrscr ;
       writeln ;
       writeln( '                   <<<  A A P M S N  >>>' ) ;
       writeln ;
       writeln ;
       writeln( 'This program uses predefined target and weapons/pattern ',
                'files to generate the' ) ;
       writeln ;
       writeln( 'input file for the main program in this series -- AAPMOD.  ',
                'First, you will be' ) ;
       writeln ;
       writeln( 'prompted for the names of the 2 files this program needs ',
                'to read.   Next, you' ) ;
       writeln ;
       writeln( 'must select the name for the file this program will ',
                'generate.   Then you will' ) ;
       writeln ;
```

```
        writeln( 'select various combinations of the input files to build ',
                 'your mission package.' ) ;
        writeln ;
        writeln( 'This program  will accept those inputs  and  produce the ',
                 'input file formatted' ) ;
        writeln ;
        writeln( 'properly for input to AAPMOD.' ) ;
        writeln ;
        pause ;
      end ;

  2 : begin
      clrscr ;
      writeln ;
      writeln( 'Crater table format is:' ) ;
      writeln ;
      writeln( 'target        wpn 1        wpn 2        wpn 3    ...' ) ;
      writeln( 'hardness' ) ;
      writeln( '   1       crater size1  crater size1  crater size1...' ) ;
      writeln( '   1       crater size2  crater size2  crater size2...' ) ;
      writeln( '   2              .             .             .' ) ;
      writeln( '   2              .             .             .' ) ;
      writeln( '   3              .             .             .' ) ;
      writeln( '   3              .             .             .' ) ;
      writeln( '   .              .             .             .' ) ;
      writeln( '   .              .             .             .' ) ;
      writeln ;
      writeln( 'Crater  size1  is the size for the weapon against  TOL ',
               'surfaces if the hardness' ) ;
      writeln( 'code is for pavements,  or  near miss  crater size ',
               'for buildings.  Crater size2' ) ;
      writeln( 'is  the  size  for  the  weapon against taxiways  ',
               'if the  hardness  code is for' ) ;
      writeln( 'pavements, or direct hits against buildings.' ) ;
      writeln ;
      pause ;
      writeln ;
    end ;

   3 : begin
        clrscr ;
        writeln ;
        writeln( 'Partial recap;  confirm the following are correct.  If ',
                 'there are errors,' ) ;
        writeln( 'you may reenter the information again.' ) ;
        writeln ;
      end ;
```

```
  4 : begin
       writeln( 'The flag to suppress  execution  of  AAPMOD  routine  ',
                'OVLAP is set,  normally' ) ;
       writeln( 'indicating  at least one pavement wider than 899.   This ',
                'routine searches for' ) ;
       writeln( 'overlapping areas of craters,  and  would  take  excessive ',
                'execution time for' ) ;
       writeln( 'the large width targets.   Note that the search is ',
                'suppressed for ALL targets' ) ;
       writeln( 'with this flag set.  However,  you may choose to override ',
                'this flag and force' ) ;
       writeln( 'AAPMOD to execute routine OVLAP.' ) ;
       writeln ;
     end ;

  5 : begin
       writeln( 'The flag to suppress execution of  AAPMOD  routine OVLAP ',
                'is not set, normally' ) ;
       writeln( 'indicating no targets wider than 899.  This  routine ',
                'searches for overlapping' ) ;
       writeln( 'areas of craters  and  takes additional execution time, ',
                'especially for larger' ) ;
       writeln( 'width targets.   You may choose  to  override this flag ',
                'and force  AAPMOD  to' ) ;
       writeln( 'suppress execution of routine OVLAP,  thus saving ',
                'execution time.   Note that' ) ;
       writeln( 'the search will be suppressed for ALL targets if you ',
                'choose to set this flag.' ) ;
       writeln ;
     end ;

  end ; { case }

end ;  { procedure info }

{==========================================================================}
procedure tgtmatrix ;
{ print target matrix data }
  var i, j : integer ;
begin
  clrscr ;
  for i := 1 to nelt do begin
    if i mod 10 = 1 then begin
      writeln ;
      writeln( 'TGT     X        Y      AXIS                    TGT  HARD',
               '  TGT    CRIT DISTANCES' ) ;
      writeln( ' #    COORD   COORD   (DEG)  LENGTH   WIDTH  TYPE  CODE',
               '  GROUP  LENGTH    WIDTH' ) ;
      writeln( '---   -----   -----   -----  ------   -----  ----  ----',
               '  -----  ------    -----' ) ;
    end ;
    write( i:3 ) ;
    for j := 1 to 5 do write( tgt[i,j]:8:1 ) ;
```

```
    if itgt[i,1] = 0
      then write( '  BLDG' )
      else if crit[i,1] < 1
        then write( '   TWY' )
        else write( '   TJL' ) ;
    write( itgt[i,2]:5, itgt[i,3]:6 ) ;
    if itgt[i,1] = 1 then write( crit[i,1]:10:1, crit[i,2]:9:1 ) ;
    writeln ;
    if i mod 10 = 0 then pause ;
  end ;
  if i mod 10 <> 0 then pause ;
end ; { procedure tgtmatrix }

{==================================================================}
procedure wpnmatrix ; { display weapons matrix on screen }
  var i, j : integer ;
      msg : string[22] ;

  procedure writeheader ;  { header for weapons matrix }
  begin
    write( 'Weapon #   !   j =  1' ) ;
    for j := 2 to nwpn do write( '        ', j ) ;
    writeln ;
    writeln( '-----------!---------------------------------------------',
             '------------' ) ;
  end ; { procedure writeheader }

begin { wpnmatrix }
  info( 2 ) ;                                                   if nhard > 8
    then msg := '   PRESS ANY KEY      '
    else msg := '                      ' ;
  writeheader ;
  writeln( 'Hardness   !   Radius of Circular Crater' ) ;
  writeln( '  Code   . !' ) ;
  for i := 1 to nhard do begin
    write( msg[ ( 2 * i ) - 1 ],' ', i:2, '  k=1 !' ) ;
    for j := 1 to nwpn do write( 2.0 / sqrt( pi ) * crtab[i,j,1]:9:1 ) ;
    writeln ;
    write( msg[ 2 * i ],'      k=2 !' ) ;
    for j := 1 to nwpn do write( 2.0 / sqrt( pi ) * crtab[i,j,2]:9:1 ) ;
    writeln ;
    if i = 9 then repeat until keypressed ;
  end ;
  writeln( '-----------!---------------------------------------------',
           '------------' ) ;
  if i > 8 then writeheader ;
  pause ;
end ; { procedure wpnmatrix }
```

```
{=================================================================}
procedure pattmatrix ;  { Display the pattern matrix on screen. }
  var i, j : integer ;
begin
  for i := 1 to npatt do begin
    writeln ;
    write( 'Pattern #', i, ' has ', ipat[i,1], ' ' ) ;

    case ipat[i,4] of

      0 : begin
            write( 'general purpose bomb' ) ;
            if ipat[i,1] <> 1 then write( 's' ) ;
          end ;

      1 : write( 'CBU ( rectangular footprint ) with ', ipat[i,2],
                 ' sub-munitions each' ) ;

      2 : write( 'CBU ( elliptical footprint ) with ', ipat[i,2],
                 ' sub-munitions each' ) ;

      3 : begin
            write( 'guided bomb' ) ;
            if ipat[i,1] <> 1 then write( 's' ) ;
          end ;

    end ; { case }

    writeln( '.' ) ;
    writeln( 'Crater Table Index:  j = ', ipat[i,3], '.  NOTE:  Error ',
             'values shown below are REP and DEP.' ) ;
    writeln ;
    if ipat[i,4] > 2
      then write( '  Optimal         Optimal       Near-miss       ',
                  'Near-Miss      ' )
      else write( ' Delivery        Delivery      Ballistic       ',
                  'Ballistic      ' ) ;
    if ipat[i,4] in [ 1, 2 ]
      then writeln( ' Cannister ' )
      else writeln( '  Weapon' ) ;
    writeln( 'Range Error   Deflection Error   Range Error     ',
             'Deflection Error   Reliability' ) ;
    writeln( '-----------   ----------------   -----------     ',
             '----------------   -----------' ) ;
    write( 0.675 * patt[i,1]:7:0 ) ;
    for j := 2 to 4 do write( 0.675 * patt[i,j]:17:0 ) ;
    writeln( patt[i,9]:17:3 ) ;
    writeln ;
    if ipat[i,4] in [ 1, 2 ] then begin
      writeln( '  Footprint        Footprint        Void           ',
               'Void           Bomblet' ) ;
      writeln( '   Length           Width           Length         ',
               'Width         Reliability' ) ;
```

```
      writeln( ' ---------        ---------        ------        ',
               '-----         -----------' ) ;
      write( 2 X patt[i,5]:8:8, 2 X patt[i,6]:18:8 ) ;
      if ipat[i,4] = 1
      then write( '          - N/A           N/A ' )
      else write( 2 X patt[i,7]:16:8, 2 X patt[i,8]:16:8 ) ;
      writeln( patt[i,10]:17:3 ) ;
    end ;
    if ipat[i,4] = 3 then begin
      writeln( 'Gross Error    Gross Error       Probability of     ',
               'Cumulative Probability of' ) ;
      writeln( 'Range Error   Deflection Error   Optimal Guidance      ',
               'Near-Miss or better' ) ;
      writeln( '-----------   ----------------   ----------------    ',
               '-------------------------' ) ;
      writeln( 0.675 X patt[i,5]:8:8, 0.675 X patt[i,6]:15:8,
               patt[i,7]:22:3, patt[i,8]:24:3 ) ;
    end ;
    if ipat[i,1] > 1 then begin
      writeln ;
      writeln ;
      writeln( 'Stick Impact Point Data:  ( Distance from mean impact ',
               'point. )' ) ;
      writeln ;
      write( ' Weapon #  1' ) ;
      for j := 2 to ipat[i,1] do write( j:6 ) ;
      writeln ;
      write( '         -----' ) ;
      for j := 2 to ipat[i,1] do write( ' -----' ) ;
      writeln ;
      write( '  Range' ) ;
      for j := 1 to ipat[i,1] do write( patt[ i, 2 X j + 9 ]:6:0 ) ;
      writeln;
      write( ' Offset' ) ;
      for j := 1 to ipat[i,1] do write( patt[ i, 2 X j + 10 ]:6:0 ) ;
      writeln ;
    end ;
    pause ;
  end ;
end ;  { procedure pattmatrix }

{===================================================================}
procedure passinfo( i : integer ) ;  { write pass information }
begin
  writeln ;
  writeln( 'Pass number ',i,' of ', npass, ' total; flown by A/C number ',
          opt[i,2], ':' ) ;
  writeln ;
end ;  { procedure passinfo }
```

```
{==================================================================}
procedure passplan ;  { display aircraft and pass sequence }
  var i, j, k : integer ;

  procedure none( n : integer ) ;
    var m : integer ;
  begin
    for m := 1 to 2 do if acft[n,m] = -1
      then write( '         none' )
      else write( acft[n,m]:12, ' ' ) ;
  end ;  { procedure none }

begin  { passplan }
  write( 'AIRCRAFT     1ST PASS     2ND PASS' ) ;
  if nacft > 16 then write('          AIRCRAFT     1ST PASS     2ND PASS' ) ;
  writeln ;
  write( '--------     --------     --------' ) ;
  if nacft > 16 then write('          --------     --------     --------' ) ;
  writeln ;
  for i := 1 to nacft do begin
    for j := 1 to 2 do acft[i,j] := -1 ;
    for j := 1 to npass do
      if opt[j,2] = i then
        if acft[i,1] = -1
          then acft[i,1] := j
          else if acft[i,1] < j
            then acft[i,2] := j
            else begin
              acft[i,2] := acft[i,1] ;
              acft[i,1] := j ;
            end ;
  end ;
  for i := 1 to 16 do
    if i <= nacft then begin
      write( i:5, ' ' ) ;
      none( i ) ;
      k := i + 16 ;
      if k <= nacft then begin
        write( k:17, ' ' ) ;
        none( k ) ;
      end ;
      writeln ;
    end ;
end ;  { procedure passplan }
```

```
{==============================================================}
procedure review ;  { see if user wants to review database matrices }
  var ichc : integer ;
begin
  writeln ;
  write( 'Would you like to review the target matrix?' ) ;
  query( ichc ) ;
  if ichc = 0 then tgtmatrix ;
  writeln ;
  write( 'Would you like to review the weapons matrix?' ) ;
  query( ichc ) ;
  if ichc = 0 then wpnmatrix ;
  writeln ;
  write( 'Would you like to review the pattern matrix?' ) ;
  query( ichc ) ;
  clrscr ;
  if ichc = 0 then pattmatrix ;
  if passflag then begin
    writeln ;
    write( 'Would you like to review the flight line-up by aircraft ',
           'number?' ) ;
    query( ichc ) ;
    clrscr ;
    if ichc = 0 then begin
      writeln ;
      passplan ;
      pause ;
    end ;
  end ;
end ;  { procedure review }

{==============================================================}
procedure init ;
{ This procedure loads in pre-exisiting target and weapon/pattern files. }
  var abortmsg, goodfile : boolean ;
      i, ichc, j, m, n : integer ;
begin
  abortmsg := false ;
  repeat processfile( fname1, 'Target' ) until filexists( fname1 ) ;
  repeat processfile( fname2, 'Weapon/Pattern' ) until filexists( fname2 ) ;
  repeat
    processfile( fname3, 'New Output' ) ;
    assign( diskfile, fname3 ) ;
    {$I-} reset( diskfile ) ; {$I+  see if file already exists }
    goodfile := ioresult <> 0 ;  { want the file not to exist }
    if not goodfile then begin
      write( 'File ( ', fname3, ' ) already exists.  Erase?' ) ;
      query( ichc ) ;
      if ichc = 0 then begin
        write( 'Are you SURE you want to erase file ( ',fname3, ' ) ?' ) ;
        query( ichc ) ;
        goodfile := ichc = 0 ;
        if goodfile then erase( diskfile ) ;
```

```
    end ;
  end ;
until goodfile ;
writeln ;
writeln( 'R E A D I N G   I N P U T   F I L E S . . .' ) ;
writeln ;
assign( diskfile, fname1 ) ;  {read in target file}
reset( diskfile ) ;
read( diskfile, nelt, ntgps, npprcw, narea ) ;
for i := 1 to nelt do begin
  for j := 1 to 5 do read( diskfile, tgt[i,j] ) ;
  for j := 1 to 3 do read( diskfile, itgt[i,j] ) ;
  if itgt[i,1] = 1 then read( diskfile, crit[i,1], crit[i,2] ) ;
end ;
read( diskfile, ncp, lv ) ;
read( diskfile, nhard, nhardp ) ;
read( diskfile, mxptch, irepr ) ;
close( diskfile ) ;
assign( diskfile, fname2 ) ; { read in weapon/pattern file }
reset( diskfile ) ;
read( diskfile, m, n, nwpn ) ;
if m <> nhard then begin
  write( 'Discrepancy in the total number of hardness levels.  You ',
          'defined ', nhard,' level' ) ;
  if nhard <> 1 then write( 's' ) ;
  writeln( ' for' ) ;
  write( 'targets, but crater sizes in terms of ', m, ' level' ) ;
  if m <> 1 then write( 's' ) ;
  writeln( '.' ) ;
  writeln ;
  abortmsg := true ;
  nhard := m ;
end ;
if n <> nhardp then begin
  write( 'Discrepancy in pavement hardness levels.  You defined ', nhardp,
          ' level' ) ;
  if nhardp <> 1 then write( 's' ) ;
  writeln( ' for' ) ;
  write( 'pavements, but pavement crater sizes in terms of ', n,' level' ) ;
  if n <> 1 then write( 's' ) ;
  writeln( '.' ) ;
  writeln ;
  abortmsg := true ;
  nhardp := n ;
end ;
for i := 1 to nhard do begin
  for j := 1 to nwpn do read( diskfile, crtab[i,j,1] ) ;
  for j := 1 to nwpn do read( diskfile, crtab[i,j,2] ) ;
end ;
```

```
  read( diskfile, npatt ) ;
   for i := 1 to npatt do begin
     for j := 1 to 4 do read( diskfile, ipat[i,j] ) ;
     for j := 1 to 10 do read( diskfile, patt[i,j] ) ;
     for j := 11 to 2 * ipat[i,1] + 10 do read( diskfile, patt[i,j] ) ;
  end ;
  close( diskfile ) ;
  if abortmsg then begin
    writeln( 'You may have to correct before proceeding.  To ABORT, press ',
             'control-C.' ) ;
    pause ;
  end ;
end ;  { procedure init }

{=================================================================}
procedure mission ;
{ This procedure collects input options. }
  var badval, goodval : boolean ;
      correct, i, ichc, j, nac, p : integer ;
      rat : real ;
begin
  writeln ;
  passflag := false ;  { suppress pass review info }
  review ;
  writeln ;
  writeln( 'Now you will build the mission package which  AAPMSN will use ',
           'to generate the' ) ;
  writeln( 'AAPMOD input file.' ) ;
  writeln ;
  badval := false ;
  repeat
    writeln ;
    if narea = 1
      then info( 4 )
      else info( 5 ) ;
    write( 'Do you want to change this flag?' ) ;
    query( ichc ) ;
    clrscr ;
    if ichc = 0 then repeat
      narea := -1 ;
      writeln ;
      writeln( 'Options for setting the OVLAP flag:' ) ;
      writeln ;
      writeln( '                 0: Enable execution of routine OVLAP.' ) ;
      writeln( '                 1: Suppress execution of routine OVLAP.' ) ;
      write( 'Enter choice ==>  ' ) ;
      intread( narea ) ;
      clrscr ;
    until narea in [ 0..1 ] ;
    writeln ;
```

```
write( 'Do you want to specify your own random number seed?' ) ;
query( ichc ) ;
if ichc = 0
  then repeat
     write( 'Enter seed  ==>  ' ) ;
     readln( seed ) ;
     repeat                    { delete leading blanks from seed }
       p := pos( ' ', seed ) ;
       if p = 1 then delete( seed, p, 1 ) ;
     until p <> 1 ;
     repeat                    { delete trailing blanks from seed }
       p := pos( ' ', seed ) ;
       if p <> 0 then delete( seed, p, 1 ) ;
     until p = 0 ;
     write( 'Confirm seed  =  ', seed ) ;
     query( ichc ) ;
    until ichc = 0
  else seed := '987654567' ;
writeln ;
write( 'Enter the desired total number of Monte Carlo samples  ==>  ' ) ;
nsamp := 200 ; { default }
intread( nsamp ) ;
writeln ;
writeln( 'Enter the desired interval between intermediate reports for ',
        'AARMOD output.' ) ;
write( '( Enter "20" for a report every 20 Monte Carlo iterations. )',
        '  ==>  ' ) ;
nsampt := 200 ; { default }
intread( nsampt ) ;
writeln ;
nflag3 := 0 ;
if nsamp > 200 then begin
  write( 'Do you want AARMOD to optimize its sampling procedures?' ) ;
  query( ichc ) ;
  if ichc = 0 then nflag3 := 1 ;
  writeln ;
end ;
clrscr ;
writeln ;
repeat
  writeln( 'What is the desired level of significance for this analysis ?' );
  write( '( 1 - confidence level )  ==>  ' ) ;
  error := 0.05 ; { default }
  realread( error ) ;
until probchk( error ) ;
writeln ;
writeln( 'What is the "Z" or Standard Normal test statistic for your ',
        'level of' ) ;
writeln( 'significance ?   Typical values:' ) ;
writeln ;
writeln( '      Confidence Level:  0.90     0.95     0.975    0.99     ',
        '0.995    0.999' ) ;
```

```
writeln( '        Significance:  0.10    0.05    0.025    0.01     ',
        '0.005   0.001' ) ;
writeln( '               Z value:  1.282    1.645    1.960    2.326     ',
        '2.576   3.090' ) ;
writeln ;
write( 'Enter test statistic  =>  ' ) ;
zalpha := 1.645 ; { default }
realread( zalpha ) ;
info( 3 ) ;
write( 'AAPMOD will ' ) ;
if narea = 1
  then write( 'suppress ' )
  else write( 'enable ' ) ;
writeln( 'execution of routine OVLAP' ) ;
writeln ;
writeln( 'Seed = ', seed, '  ( Default = 987654567 )' ) ;
writeln ;
writeln( 'Number of Monte Carlo samples = ', nsamp ) ;
writeln ;
if nsamp > 200 then begin
  write( 'AAPMOD will ' ) ;
  if nflag3 = 0 then write( 'not ' ) ;
  writeln( 'optimize its sampling procedures.' ) ;
  writeln ;
end ;
writeln( 'Report interval = ', nsampt ) ;
writeln ;
writeln( 'Level of significance = ', error:3:4 ) ;
writeln ;
writeln( 'Standard Normal test statistic = ', zalpha:3:4 ) ;
writeln ;
write( 'Are the above correct?' ) ;
query( correct ) ;
clrscr ;
writeln ;
if correct = 1 then begin
  writeln( 'Reenter the following: ' ) ;
  writeln ;
end
until correct = 0 ;
repeat
  badval := false ;
  writeln( 'Enter the following:' ) ;
  npass := -1 ;
  repeat
    writeln ;
    write( 'Number of passes over the target complex ( max 32 )  =>  ' ) ;
    intread( npass ) ;
  until chkgood( npass, 32, 1 ) ;
```

```
nac := 0 ;
repeat
  writeln ;
  writeln( 'Each aircraft may reattack one time.' ) ;
  writeln ;
  write( 'Number of aircraft participating in the attack  =>  ' ) ;
  intread( nac ) ;
  nacft := nac ;
  writeln ;
  if nac < 1  { trap for 0 or negative acft }
    then rat := 3
    else rat := npass / nac ;
  if rat > 2 then writeln( 'Insufficient A/C to accomplish attack.' ) ;
until ( rat <= 2 ) and chkgood( nac, npass, ( npass + 1 ) div 2 ) ;
kac := 0 ; { aircraft count }
for i := 1 to npass do begin  { initialize arrays }
  for j := 1 to 4 do pass[i,j] := 0 ;
  pass[i,5] := 1 ;  { reattack probability }
  for j := 1 to 2 do begin
    opt[i,j] := 0 ;
    ipass[i,j] := 0 ;
  end ;
end ;
passflag := true ;  { enable review of pass info }
i := 1 ;
repeat { i - loop }
  clrscr ;
  writeln ;
  write( 'Pass number ',i,' of ', npass, ' total; ' ) ;
  if opt[i,1] = 0 then begin { 1st pass for acft }
    p := 0 ; { p will count # of passes not yet assigned to acft }
    for j := 1 to npass do if opt[j,2] = 0 then p := p + 1 ;
    if ( nac - kac ) * 2 < p then begin  { not enough acft }
      badval := true ;
      writeln ;
      writeln ;
      writeln( 'E R R O R :  Insufficient aircraft available.' ) ;
      writeln ;
      writeln( '            Total number of aircraft for this mission = ',
               nac ) ;
      writeln( '            Current number of aircraft assigned to ',
               'passes = ', kac ) ;
      writeln ;
      writeln( '            Number of passes without aircraft assigned = ',
               p ) ;
      writeln( '            Number of aircraft available for unassigned ',
               'passes = ', nac - kac ) ;
      writeln ;
      writeln( 'To correct this error condition, you will have to ',
               're-enter all pass data.' ) ;
      writeln( 'You may ABORT the program by pressing control-C.' ) ;
      writeln ;
      writeln ;
```

```
      write( 'Would you like to review the flight line-up by aircraft ',
             'number?' ) ;
      query( ichc ) ;
      clrscr ;
      if ichc = 0 then begin
        writeln ;
        writeln( 'Flight line-up that caused the error condition:' ) ;
        writeln ;
        passplan ;
        pause ;
      end ;
      writeln ;
    end ;
    kac := kac + 1 ;  { need new acft }
  end ;
  if not badval then begin  { continue loop }
    repeat
      if opt[i,1] = 0 then opt[i,2] := kac ;  { assign acft to this pass }
      writeln( 'flown by A/C number ', opt[i,2], ':' ) ;
      writeln ;
      review ;
      passinfo( i ) ;
      write( 'Enter element number of primary target  ==>  ' ) ;
      intread( npx[i] ) ;
      writeln ;
      write( 'Preplanned aimpoint -- x-coord  ==>  ' ) ;
      realread( pass[i,1] ) ;
      writeln ;
      write( 'Preplanned aimpoint -- y-coord  ==>  ' ) ;
      realread( pass[i,2] ) ;
      writeln ;
      write( 'Attack direction ( referenced CCW from +x-axis )  ==>  ' ) ;
      realread( pass[i,3] ) ;
      repeat
        writeln ;
        writeln( 'Next you must enter the attack pattern ( from the ',
                 'pattern matrix ) this' ) ;
        write( 'aircraft will fly.  Do you want to review the pattern ',
               'matrix?' ) ;
        query( ichc ) ;
        clrscr ;
        if ichc = 0 then pattmatrix ;
        passinfo( i ) ;
        write( 'Attack pattern code from pattern matrix  ==>  ' ) ;
        intread( ipass[i,1] ) ;
        if ( ipass[i,1] > npatt ) or ( ipass[i,1] <= 0 ) then begin
          writeln( 'Undefined pattern.  If irreconciliable at this point ',
                   'you must terminate' ) ;
          writeln( 'and define the pattern with AAPWPN.  For the ',
                   'weapons file currently in' ) ;
          writeln( 'use ( ', fname2, ' ), total number of defined patterns ',
                   '= ', npatt, '.' ) ;
          writeln ;
```

```
      writeln( 'To ABORT program, press control-C.' ) ;
    end ;
  until chkgood( ipass[i,1], npatt, 1) ;
  info( 3 ) ;
  passinfo( i ) ;
  writeln( 'Element number of primary target  =  ', npx[i] ) ;
  writeln ;
  writeln( 'Preplanned aimpoint -- x-coord  =  ', pass[i,1]:3:1 ) ;
  writeln ;
  writeln( 'Preplanned aimpoint -- y-coord  =  ', pass[i,2]:3:1 ) ;
  writeln ;
  writeln( 'Attack direction ( referenced CCW from +x-axis )  =  ',
          pass[i,3]:3:1 ) ;
  writeln ;
  writeln( 'Attack pattern code from pattern matrix  =  ',
          ipass[i,1] ) ;
  writeln ;
  writeln ;
  write( 'Is this information correct ?' ) ;
  query( correct ) ;
  clrscr ;
  if correct = 1 then begin
    writeln ;
    write( 'Reenter pass number ', i, ' of ', npass, ' total; ' ) ;
  end ;
until correct = 0 ;
if opt[i,1] = 0
  then begin
      pass[i,4] := -1 ;
      repeat
        passinfo( i ) ;
        writeln( 'Enter the probability that the aircraft survives ',
               'enroute attrition and' ) ;
        write( 'begins its first pass.  ==>  ' ) ;
        realread( pass[i,4] ) ;
      until probchk( pass[i,4] ) ;
      goodval := true ;
      if i < npass
        then repeat
          clrscr ;
          writeln ;
          if goodval
            then begin
                writeln( 'If this aircraft will fly a second pass, ',
                        'enter which of the remaining passes' ) ;
                writeln( '( out of ', npass, ' total ) it will fly.  ',
                        'Enter 0 for no second pass.' ) ;
              end
            else writeln( 'Pass ', ipass[i,2], ' is not available.' ) ;
          writeln ;
          passplan ;
          writeln ;
          write( 'Second pass will be number  ==>  ' ) ;
```

```
              intread( ipass[i,2] ) ;
              clrscr ;
              goodval := ( ipass[i,2] <= npass ) and ( ipass[i,2] >= 0 ) ;
              if goodval and ( ipass[i,2] <> 0 ) {array subscript in range}
                then if ( opt[ ipass[i,2],2 ] > 0 )
                  then goodval := false ;
            until goodval
            else ipass[i,2] := 0 ;
          if ipass[i,2] > 0 then begin  { for next pass info: }
            pass[i,5] := -1 ;
            repeat
              writeln ;
              writeln( 'Enter the probability that the aircraft survives ',
                       'target area attrition and' ) ;
              write( 'begins its second pass.  ==>  ' ) ;
              realread( pass[i,5] ) ;
            until probchk( pass[i,5] ) ;
            opt[ ipass[i,2],1 ] := 1 ;    { second pass flag }
            opt[ ipass[i,2],2 ] := kac ;  { assign acft to second pass }
          end ;
        end
      else begin  { current pass is acft's second pass }
          pass[i,4] := 1 ;  { probability of enroute attrition: N/A }
          ipass[i,2] := 0 ; { can't have a third pass }
          pass[i,5] := 1 ;  { probability of survival to reattack: N/A }
        end ;
    i := i + 1 ;  { pass counter }
   end ;
  until ( i > npass ) or badval ;  { loop terminates normally at npass + 1 }
 until not badval ;   {  loop in case of discrepancy in number of A/C }
end ;  { procedure mission }
```

```
{=========================================================================}
procedure outdat ;
{ This procedure writes the input file for AAPMOD. }
  var i, j, jr : integer ;
begin
  clrscr ;
  writeln ;
  writeln ;
  writeln( 'Now you are ready to run the main program  --  AAPMOD.   ',
           'Remember the file' ) ;
  writeln ;
  writeln( 'name shown below, as you will need to enter it when prompted by ',
           'AAPMOD.' ) ;
  writeln ;
  writeln ;
  writeln ;
  writeln( 'Writing AAPMOD input file to disk file:   < ', fname3, ' >' ) ;
  assign( diskfile, fname3 ) ;
  rewrite( diskfile ) ;
  writeln( diskfile, seed:10 ) ;
  writeln( diskfile, nsamp:5, nsampt:5 ) ;
  writeln( diskfile, nflag3:3, error:6:2, zalpha:7:3 ) ;
  writeln( diskfile, nelt:5, ntgps:5, npprcw:5, narea:5 ) ;
  for i := 1 to nelt do begin
    for j := 1 to 5 do write( diskfile, tgt[i,j]:9:1 ) ;
    for j := 1 to 3 do write( diskfile, itgt[i,j]:5 ) ;
    writeln( diskfile ) ;
    if itgt[i,1] = 1 then writeln( diskfile, crit[i,1]:9:1, crit[i,2]:9:1 )
  end ;
  writeln( diskfile, ncp:5, lv:5 ) ;
  writeln( diskfile, npatt:5 ) ;
  for i := 1 to npatt do begin
    for j := 1 to 4 do write( diskfile, ipatt[i,j]:5 ) ;
    writeln( diskfile ) ;
    for j := 1 to 8 do write( diskfile, patt[i,j]:8:1 ) ;
    writeln( diskfile, patt[i,9]:7:3, patt[i,10]:7:3 ) ;
    for j := 1 to ipatt[i,1] do begin
      jr := 2 * j + 9 ;
      writeln( diskfile, patt[i,jr]:8:1, patt[i, jr + 1]:8:1 ) ;
    end ;
  end ;
  writeln( diskfile, nhard:5, nwpn:5 ) ;
  for i := 1 to nhard do begin
    for j := 1 to nwpn do write( diskfile, crtab[i,j,1]:7:1 ) ;
    writeln( diskfile );
    for j := 1 to nwpn do write( diskfile, crtab[i,j,2]:7:1 ) ;
    writeln( diskfile );
  end ;
  writeln( diskfile, mxptch:5, irepr:5, npass:5 ) ;
```

```
for i := 1 to npass do begin
  for j := 1 to 3 do write( diskfile, pass[i,j]:9:1 ) ;
  for j := 4 to 5 do write( diskfile, pass[i,j]:8:3 ) ;
  writeln( diskfile, ipass[i,1]:5, ipass[i,2]:5, npx[i]:5 ) ;
end ;
close( diskfile ) ;
writeln ;
writeln ;
writeln ;
end ;  { procedure outdat }

{====================== M A I N   P R O G R A M ======================}

begin

info( 1 ) ;
init ;
mission ;
outdat ;
writeln( 'E N D   O F   P R O G R A M' ) ;

end.
```

APPENDIX C

FORTRAN LISTING FOR THE ATTACK ASSESSMENT PROGRAM-MODIFIED

This appendix provides the source code listing for the FORTRAN V version of the Attack Assessment Program-MODIFIED (AAPMOD). The program, as shown, was written for the Control Data Corporation (CDC) 6600 CYBER.

```
*****************************************************************
* LAST UPDATE 31/1500 JAN 85          FILE:MAIN.AAPMOD           *
*****************************************************************
      PROGRAM AAPMOD
*****************************************************************
*                 AIRFIELD ATTACK PROGRAM                        *
*                                                                *
*--USED AT EGLIN AFB, FL, AND 50-60 CONTRACTOR LOCATIONS.         *
*  DEVELOPED AT OKLAHOMA STATE UNIVERSITY,                        *
*  UNDER CONTRACT F08635-79-C-0255, FOR THE JOINT TECHNICAL COORDINATING
*  GROUP FOR MUNITIONS EFFECTIVENESS.                             *
*  MODIFIED BY CAPTAIN ROBERT N. MIGLIN TO PROVIDE INTERACTIVE    *
*  CAPABILITY FOR TACTICS ASSESSMENT.                             *
*  FURTHER UPDATES BY MAJOR DAVID A. ROODHOUSE AND CAPTAIN T.K. GREEN *
*                                                                *
*--NON-ANSI.  ANSI=0 REQUIRED FOR CDC 6600.                       *
*                                                                *
*****************************************************************
*
      CHARACTER FNAME*6,FNAME2*6
      INTEGER NPX(32)
*
      COMMON
     1 ADM(112)       ,GPHT(15)      ,MXPTCH        ,SIGADM(112),
     2 AMIN(3)        ,GPHTAC(15)                   ,SIGARP(3),
     3 APRA(3)        ,GPHTS(15)                    ,SIGASP(3),
     4 APRMIN(3)      ,LNHITS(112)   ,NSAMP1        ,SIGCRT(3),
     5 AREP(3)        ,ICRAT(4)      ,PASS(0:32,6)  ,SIGCTS(27),
     6 ASTP(3)        ,ICUT(4,3)     ,PATT(13,34)   ,SIGFIL(27),
     7 COUNTR(112)    ,IHIT(3)       ,RAPF(112)     ,SIGHTS(112),
     8 CRIT(112,2)    ,IPASS(32,2)   ,RCUT(112)     ,SIGNAF(112),
     9 CRTAB(11,6,2)  ,IPAT(12,4)    ,RHIT(112)     ,SMINA(4),
     & DECAR(112)     ,IPCUT(3)      ,SAPR(4)       ,SNAPFL(3),
     1 DSTR(3)                       ,SAPRA(4)      ,TGT(112,5),
     2 ENAPFL(3)      ,IPL(40)       ,SAVE(800,3)   ,XC(3),
     3 GPADAC(15)     ,ISAV(800)     ,SGAPR(4)      ,YC(3),
     4 GPADM(15)      ,ITGT(112,3)   ,SGAPRA(4),
     5 GPADMS(15)     ,I2CUT(4)      ,SGCRAT(4),
     6 GPAREA(15)     ,KH(3)         ,SGMINA(4)
*
      COMMON/RAY/TWOPI
*
      COMMON/RAY2/SQUARE(900),CRMAX
*
      COMMON/END/NSAMP2,NELT,NTGPS,NCP,CRMIN,APPROM,NAREA
*
      COMMON/CATA/NUMAPR(4,2),STEMP(303)
*
      COMMON/JOHN/NFLAG1,NFLAG2,NMAX,NSAMPR,ZALPH,ERROR,NSAMP,NFLAG3
```

```
*
*-----INPUT/INITIALIZE
*
      TWOPI=6.28318530718
      SRPI04=0.8862269254528
      POV180=0.01745329252
*
      PRINT*,'NAME OF INPUT FILE==> '
      READ 901,FNAME
      OPEN(UNIT=12,FILE=FNAME,STATUS='OLD')
      REWIND 12
      PRINT*,'NAME OF OUTPUT FILE==> '
      READ 901,FNAME2
      OPEN(UNIT=13,FILE=FNAME2,STATUS='NEW')
*
10    READ(12,991)ISEED
      CALL RANSET(ISEED)
      ITDI=1
      READ(12,*)NSAMP,NSAMPT
      READ(12,*)NFLAG3,ERROR,ZALPH
*
*-----READ TARGET DESCRIPTION
*
      READ(12,*)NELT,NTGPS,APPRCH,NAREA
      DO 30 I=1,NELT
       READ(12,*)(TGT(I,J),J=1,5),(ITGT(I,J),J=1,3)
       TGT(I,3) = TGT(I,3)*0.0174533
       CRIT(I,1)=0.
       CRIT(I,2)=0.
       IF (ITGT(I,1).EQ.1) THEN
         READ(12,*)CRIT(I,1),CRIT(I,2)
       ENDIF
30    CONTINUE
      READ(12,*)NCP,LV
*
*-----READ PATTERN DESCRIPTIONS
*
      READ(12,*)NPATT
      DO 40 I=1,NPATT
       READ(12,*)(IPAT(I,J),J=1,4)
       READ(12,*)(PATT(I,J),J=1,10)
       DO 40 J=1,IPAT(I,1)
        JR=2*J+9
        READ(12,*)PATT(I,JR),PATT(I,JR+1)
40    CONTINUE
*
*-----READ CRATERING TABLE
*
      READ(12,*)M,N
      DO 52 I=1,M
       READ(12,*)(CRTAB(I,J,1),J=1,N)
       READ(12,*)(CRTAB(I,J,2),J=1,N)
52    CONTINUE
```

```
      CRMIN=1.0E10
      CRMAX=0.
*
*-----READ MISSION DESCRIPTION
*  IREPR... TELLS WHAT TYPE OF REPAIRS ARE TO BE MADE
*                 = 0--ALL MAJOR PAVEMENTS (CRIT(L,1)>0)
*                      ARE REPAIRED IN ORDER INPUT
*                 = 1--EASIEST STRIP TO REPAIR FIXED FIRST,
*                      THEN REST WITH (CRIT(L,1)>0) IN ORDER INPUT
*                 = 2--ONLY EASIEST STRIP TO REPAIR IS DONE
*                 =1X--REPAIR STRIP AND APPROACH IN ORDER OF
*                      "X" ABOVE, I.E., 11 => APPROACHES AND 1.
*
      READ(12,*)MXPTCH,IREPR,NPASS
      DO 70 I=1,NPASS
       READ(12,*)(PASS(I,J),J=1,5),(IPASS(I,J),J=1,2),NPX(I)
       PASS(I,3) = PASS(I,3)*0.0174533
       PASS(I,6)=PASS(I,5)
70    CONTINUE
      CLOSE(UNIT=12)
*
      PRINT*,'DO YOU WANT AN OUTPUT ECHO OF INPUT? 1=YES, 0=NO =>'
      READ*,OECHO
      IF (OECHO.EQ.1) THEN
        WRITE(13,1970)
        WRITE(13,1950)ISEED
        WRITE(13,1950)NSAMP,NSAMPT
        WRITE(13,1980)NFLAG3,ERROR,ZALPH
        WRITE(13,1975)NELT,NTGPS,APPRCH,NAREA
        DO 1500 I=1,NELT
         WRITE(13,1955)TGT(I,1),TGT(I,2),TGT(I,3)*360/TWOPI,
     1      TGT(I,4),TGT(I,5),(ITGT(I,J),J=1,3)
         IF (ITGT(I,1).EQ.1) WRITE(13,1955)CRIT(I,1),CRIT(I,2)
1500    CONTINUE
        WRITE(13,1950)NCP,LV
        WRITE(13,1950)NPATT
        DO 1510 I=1,NPATT
         WRITE(13,1960)(IPAT(I,J),J=1,4)
         WRITE(13,1961)(PATT(I,J),J=1,10)
         DO 1510 J=1,IPAT(I,1)
          JRNG=2*J+9
          WRITE(13,1955)PATT(I,JRNG),PATT(I,JRNG+1)
1510    CONTINUE
        WRITE(13,1950)M,N
        DO 1522 I=1,M
         WRITE(13,1965)(CRTAB(I,J,1),J=1,N)
         WRITE(13,1965)(CRTAB(I,J,2),J=1,N)
1522    CONTINUE
        WRITE(13,1950)MXPTCH,IREPR,NPASS
        DO 1530 I=1,NPASS
         WRITE(13,1955)PASS(I,1),PASS(I,2),PASS(I,3)*360/TWOPI,
     1      PASS(I,4),PASS(I,5),(IPASS(I,J),J=1,2),NPX(I)
1530    CONTINUE
```

```
      ENDIF
*
*-----INITIALIZE FOR MONTE CARLO
*
      WRITE(13,905)FNAME,FNAME2
      NSAMPR=1
      DO 88 I=1,NELT
       ITGTTP=ITGT(I,2)
       DO 88 J=1,NPASS
        NPTRN=IPASS(J,1)
        JWPNTP=IPAT(NPTRN,3)
        IF (ITGT(I,1).EQ.1) THEN
          TBHLD1=CRTAB(ITGTTP,JWPNTP,1)
          TBHLD2=CRTAB(ITGTTP,JWPNTP,2)
          CRMIN=AMIN1(CRMIN,TBHLD1,TBHLD2)
          CRMAX=AMAX1(CRMAX,TBHLD1,TBHLD2)
        ENDIF
88    CONTINUE
      CALL INITL(NELT,NTGPS,NCP,LV)
      NMAX=0
*
*--TEST TO SEE IF LIMITING MONTE CARLO LOOPS IS BOTH DESIRED (NFLAG3=1)
*  AND APPROPRIATE (NSAMP)200).  IF SO, SET FLAGS AND SET INITIAL
*  MONTE CARLO LOOP LIMIT.
*
      IF ((NFLAG3.EQ.1).AND.(NSAMP.GE.200)) THEN
        NFLAG1=0
        NFLAG2=0
        NMAX=NSAMP
        NSAMP=200
      ENDIF
*
*
*-----MONTE CARLO LOOP  --  820 ON (IT/
*
85    DO 820 IT=NSAMPR,NSAMP
*
*-----INITIALIZE VARIABLES WHICH GET RESET EACH MONTE CARLO REP
*
       NSAMP2=IT
100    DO 110 L=1,NELT
        DECAR(L)=TGT(L,4)*TGT(L,5)
110    CONTINUE
       DO 120 L=1,3
        IPCUT(L)=0
        IHIT(L)=0
        AMIN(L)=0.
        APMIN(L)=0.
        APRA(L)=0.
120    CONTINUE
       M=0
       M0=0
       KZ=0
```

```
*
*------SET NUMBER OF HITS PER TARGET EQUAL TO ZERO
*
      DO 130 L=1,NELT
       LNHITS(L)=0
130   CONTINUE
*
*------COMPUTE IMPACT POINTS OF WEAPONS
*
200   DO 370 I=1,NPASS
*
*-------SEE IF A/C SURVIVED. IF YES, CHANGE NEXT PASS PS TO REATTACK PS
*                           IF NOT, CHANGE NEXT PASS PS TO 0.0,
*                                   AND LOG NO HITS FOR THIS PASS
*
       NXTP=IPASS(I,2)
       CRAZYN=RANF()
       IF (CRAZYN.GT.PASS(I,4)) THEN
         PASS(NXTP,4)=0.
         GO TO 370
       ELSE
         PASS(NXTP,4)=PASS(I,5)
       ENDIF
*
       NPTRN=IPASS(I,1)
       NWEP=IPAT(NPTRN,1)
       NBOM=IPAT(NPTRN,2)
       RMAJ=PATT(NPTRN,5)
       RMIN=PATT(NPTRN,6)
       UMAJ=PATT(NPTRN,7)
       UMIN=PATT(NPTRN,8)
       KODE=IPAT(NPTRN,4)
*
*-------LOCATE STICK PATTERN CENTER
*
       PASSXT=PASS(I,1)
       PASSYT=PASS(I,2)
      IF (NPX(I).LE.NCP) THEN
        NTTT=NPX(I)
        CALL TRISUB(DAP)
        PASSXT=PASSXT+DAP*COS(TGT(NTTT,3))
        PASSYT=PASSYT+DAP*SIN(TGT(NTTT,3))
      ENDIF
       SINP=SIN(PASS(I,3))
       COSP=COS(PASS(I,3))
```

```
210    IF (KODE.EQ.3) THEN
*- - - - -GUIDED MUNITIONS...
          CRAZYN=RANF()
          IF (CRAZYN.LE.PATT(NPTRN,7)) THEN
            CALL NORAN (R,PATT(NPTRN,1),D,PATT(NPTRN,2))
          ELSE
            IF (CRAZYN.LE.PATT(NPTRN,8)) THEN
              CALL NORAN (R,PATT(NPTRN,3),D,PATT(NPTRN,4))
            ELSE
              CALL NORAN (R,PATT(NPTRN,5),D,PATT(NPTRN,6))
            ENDIF
          ENDIF
          X=PASSXT+R*COSP+D*SINP
          Y=PASSYT+R*SINP-D*COSP
        ELSE
*- - - - -DUMB BOMBS...
          CALL NORAN (R,PATT(NPTRN,1),D,PATT(NPTRN,2))
          XCTR=PASSXT+R*COSP+D*SINP
          YCTR=PASSYT+R*SINP-D*COSP
        ENDIF
*
*-------LOCATE WEAPON IMPACT OR CENTER OF DISPENSER PATTERN
*
        DO 360 K=1,NWEP
         CRAZYN=RANF()
         IF (CRAZYN.GT.PATT(NPTRN,9)) GO TO 360
         IF (KODE.LT.3) THEN
           CALL NORAN (R,PATT(NPTRN,3),D,PATT(NPTRN,4))
           K2=2*K+9
           XIMOD=XCTR+(PATT(NPTRN,K2)+R)*COSP+(PATT(NPTRN,K2+1)+D)*SINP
           YIMOD=YCTR+(PATT(NPTRN,K2)+R)*SINP-(PATT(NPTRN,K2+1)+D)*COSP
         ENDIF
*
*--------LOCATE IMPACTS  (NBOM = 1 OR NMBR BOMBLETS/CBU SHELL)
*
270      DO 350 MI=1,NBOM
          IF (KODE.LT.3) THEN
            X=XIMOD
            Y=YIMOD
            IF (NBOM.GT.1) THEN
              CRAZYN=RANF()
              IF (CRAZYN.GT.PATT(NPTRN,10)) GO TO 350
280           CRAZYN=RANF()
              XI=2.*RMAJ*CRAZYN-RMAJ
              CRAZYN=RANF()
              YI=2.*RMIN*CRAZYN-RMIN
              IF (KODE.EQ.2) THEN
                XIYIOL=(XI**2/RMAJ**2)+(YI**2/RMIN**2)
                IF (XIYIOL.GT.1.) GO TO 280
                IF ((UMAJ.GT.0.).AND.(UMIN.GT.0.)) THEN
                  XIYIIL=(XI**2/UMAJ**2)+(YI**2/UMIN**2)
                   IF (XIYIIL.LT.1.) GO TO 280
                ENDIF
```

```
              ENDIF
290           X=X+X1*COSP+Y1*SINP
              Y=Y+X1*SINP-Y1*COSP
            ENDIF
          ENDIF
*
*---------CHECK FOR ANY HIT OR NEAR-MISS
*
300       DO 340 L=1,NELT
           SINT=SIN(TGT(L,3))
           COST=COS(TGT(L,3))
           XP=X-TGT(L,1)
           YP=Y-TGT(L,2)
           T1=XP*COST+YP*SINT
           XP=YP*COST-XP*SINT
           ITGTTP=ITGT(L,2)
           JWPNTP=IPAT(NPTRN,3)
           IF ((L.GT.NCP).AND.(L.LE.(LV+NCP))) THEN
            IF(ABS(T1)-CRTAB(ITGTTP,JWPNTP,2).GE..5*TGT(L,4)) GO TO 340
            IF(ABS(XP)-CRTAB(ITGTTP,JWPNTP,2).GE..5*TGT(L,5)) GO TO 340
           ELSE
            IF(ABS(T1)-CRTAB(ITGTTP,JWPNTP,1).GE..5*TGT(L,4)) GO TO 340
            IF(ABS(XP)-CRTAB(ITGTTP,JWPNTP,1).GE..5*TGT(L,5)) GO TO 340
           ENDIF
330        M=M+1
           IF (M.LE.800) THEN
             SAVE(M,1)=T1+.5*TGT(L,4)
             SAVE(M,2)=XP+.5*TGT(L,5)
             SAVE(M,3)=FLOAT(L)
             ISAV(M)=IPAT(NPTRN,3)
             COUNTR(L)=COUNTR(L)+1.
             LNHITS(L)=LNHITS(L)+1.
           ENDIF
340       CONTINUE
          IF (M.GT.800) WRITE(13,1200)I,M
          M=MIN0(M,800)
350      CONTINUE
360     CONTINUE
370    CONTINUE
       KIEND=0
       IF (M.EQ.0) THEN
         WRITE(13,840)IT
         GO TO 910
       ENDIF
*
*------MISSION IS COMPLETE
*
390    CALL SORT (M,SAVE(1,3),SAVE(1,1),SAVE(1,2),ISAV(1))
       DO 400 J=1,NTGPS
        GPADAC(J)=0.
        GPHTAC(J)=0.
400    CONTINUE
```

```
      DO 410 I=1,NELT
       ITGTGP=ITGT(I,3)
       GPHTAC(ITGTGP)=GPHTAC(ITGTGP)+LNHITS(I)
       SIGHTS(I)=SIGHTS(I)+LNHITS(I)**2
410   CONTINUE
      DO 420 J=1,NTGPS
       GPHTS(J)=GPHTS(J)+GPHTAC(J)**2
420   CONTINUE
*
*------INITIAL CONDITIONS FOR BDA
*
      L=1
      K0=1
      CUTS=0.
      FILL=0.
      ARFILL=0.
      ARFILS=0.
      SUMRUN=0.
*
*------LOOP ON EACH IMPACT, AND GROUP ON TARGET ELEMENT
*
      DO 740 K=1,N
       IF ((SAVE(K,3).GT.FLOAT(L)).OR.(K.EQ.N)) THEN
430     N=K-K0
        IF (SAVE(K,3).EQ.FLOAT(L)) N=N+1
*
*---------DAMAGE ASSESSMENT
*
500     IF (ITGT(L,1).EQ.0) THEN
*
*-----------TGT L IS A BUILDING OR NON-PAVEMENT
*
         IF ((N.GT.0).AND.(DECAR(L).GT.0.))
     1       CALL BLDG(SAVE(K0,1),SAVE(K0,2),CRTAB,ITGT(L,2),
     2                ISAV(K0),N,TGT(L,4),TGT(L,5),DECAR(L))
         DIFFAR=TGT(L,4)*TGT(L,5)-DECAR(L)
         ADM(L)=ADM(L)+DIFFAR
         SIGADM(L)=SIGADM(L)+DIFFAR**2
         ITGTGP=ITGT(L,3)
         GPADAC(ITGTGP)=GPADAC(ITGTGP)+DIFFAR
        ELSE
*
*---------TGT L IS A PAVEMENT
*
         IF (N.LT.1) THEN
          IF (L.LE.NCP) THEN
           XC(L)=.5*(TGT(L,4)+CRIT(L,1))
           YC(L)=.5*(TGT(L,5)-CRIT(L,2))
          ENDIF
          GO TO 730
         ENDIF
520      CALL SORT(N,SAVE(K0,1),SAVE(K0,2),SAVE(K0,3),ISAV(K0))
         IF (NAREA.EQ.0) CALL OVLAF
```

```
     1        (SAVE(K0,1),SAVE(K0,2),CRTAB,ITGT(L,2),ISAV(K0),0.,0.,
     2        IFIX(TGT(L,4)),IFIX(TGT(L,5)),N,SUMRUN)
*
*-----------TAXIWAYS (MINOR PAVEMENTS)
*         FIND WANDERING PATH ONLY FOR TAXI-ONLY TARGETS  (CRIT(L,1)=0.)
*
          IF (CRIT(L,1).LT.1.0) THEN
            CALL MINCW (CRMAX,N,SAVE(K0,1),SAVE(K0,2),
     1                  CRTAB(1,1,2),ITGT(L,2),ISAV(K0),
     2                  CRIT(L,2),TGT(L,5),NFILL,CUTS,ARFILL)
            ARFILS=ARFILL
            FILL=FLOAT(NFILL)
710         RHIT(L)=RHIT(L)+FILL
            NTXWY=L-NCP
            SIGFIL(NTXWY)=SIGFIL(NTXWY)+FILL*FILL
            RCUT(L)=RCUT(L)+CUTS
            SIGCTS(NTXWY)=SIGCTS(NTXWY)+CUTS*CUTS
          ELSE
*
*-----------RUNWAYS (MAJOR PAVEMENTS)
*   SEARCH FOR A CLEAR STRIP (LENGTH=CRIT(L,1) .X. WIDTH=CRIT(L,2))
*
            M0=K-1
            IF ((K.EQ.M).AND.(SAVE(M,3).EQ.FLOAT(L))) M0=M0+1
            IPCUT(L)=0
            IHIT(L)=0
            AMIN(L)=0.
            APRMIN(L)=0.
            APRA(L)=0.
            DO 540 KK=1,4
             DO 540 KK2=1,2
              NUMAPR(KK,KK2)=0
540         CONTINUE
            CALL CLSTRP(CRMAX,N,SAVE(K0,1),SAVE(K0,2),CRTAB,
     1                  ITGT(L,2),ISAV(K0),TGT(L,4),TGT(L,5),
     2                  CRIT(L,1),CRIT(L,2),XC(L),YC(L),NMIN)
            IF (NMIN.GT.0) THEN
              RCUT(L)=RCUT(L)+1.
              IPCUT(L)=1
            ENDIF
550         RHIT(L)=RHIT(L)+FLOAT(NMIN)
            IHIT(L)=NMIN
            SUMSTP=0.
            KM1=K-1
            IF ((K.EQ.M).AND.(SAVE(K,3).EQ.FLOAT(L))) KM1=K
            KA=1
            MFLAG=0
            XS1=XC(L)
            YS1=YC(L)
            XS2=XC(L)-CRIT(L,1)
            YS2=YC(L)+CRIT(L,2)
            KH(L)=KZ
            KP1=K0
```

```
560       CONTINUE
          IF ((KP1.LE.M).AND.(KM1.LE.M)) THEN
            ITGTTP=ITGT(L,2)
            DO 580 KW=KP1,KM1
             JWPNTP=ISAV(KW)
             IF(SAVE(KW,1)+CRTAB(ITGTTP,JWPNTP,KA).LE.XS2) GO TO 580
             IF(SAVE(KW,1)-CRMAX.GE.XS1) GO TO 590
             IF(SAVE(KW,1)-CRTAB(ITGTTP,JWPNTP,KA).GE.XS1) GO TO 580
             IF(SAVE(KW,2)+CRTAB(ITGTTP,JWPNTP,KA).LE.YS1) GO TO 580
             IF(SAVE(KW,2)-CRTAB(ITGTTP,JWPNTP,KA).GE.YS2) GO TO 580
             KZ=KZ+1
             IF (KW.NE.KZ) THEN
               S1=SAVE(KW,1)
               S2=SAVE(KW,2)
               S3=SAVE(KW,3)
               ITT=ISAV(KW)
               KZP=KZ+1
               DO 570 K8=KZP,KW
                KK=KW-K8+KZP
                SAVE(KK,1)=SAVE(KK-1,1)
                SAVE(KK,2)=SAVE(KK-1,2)
                SAVE(KK,3)=SAVE(KK-1,3)
                ISAV(KK)=ISAV(KK-1)
570            CONTINUE
               SAVE(KZ,1)=S1
               SAVE(KZ,2)=S2
               SAVE(KZ,3)=S3
               ISAV(KZ)=ITT
             ENDIF
580         CONTINUE
          ENDIF
590       IF (MFLAG.EQ.0) THEN
            KZT=0
            IF (KZ.NE.KH(L)) THEN
              KZT=KZ-KH(L)
              KK=KH(L)+1
              KH(L)=KZ
              IF (NAREA.LE.0) CALL OVLAP
     1          (SAVE(KK,1),SAVE(KK,2),CRTAB,ITGT(L,2),
     2           ISAV(KK),XC(L)-CRIT(L,1),YC(L),IFIX(CRIT(L,1)),
     3           IFIX(CRIT(L,2)),KZT,SUMSTP)
600           ASTP(L)=ASTP(L)+SUMSTP
              SIGASP(L)=SIGASP(L)+SUMSTP*SUMSTP
              AMIN(L)=SUMSTP
            ENDIF
610         MFLAG=1
            KA=2
            KP1=KP1+KZT
            KZ=KP1-1
            KZ1=KZ
            XS1=XC(L)-CRIT(L,1)
```

```
          IF (XS1.GE.CRIT(L,2)) THEN
            XS2=CRIT(L,2)
            GO TO 560
          ELSE
            GO TO 640
          ENDIF
        ENDIF
620     KZT=0
        NFILL=0
        IF (KZ.NE.KZ1) THEN
          KZT=KZ-KZ1
          KK=KZ1+1
          IF (MFLAG.GE.3) CALL SORT
     1       (KZT,SAVE(KK,2),SAVE(KK,1),SAVE(KK,3),ISAV(KK))
          DO 892 II=1,KZT
           SAVE(KK+II-1,2)=SAVE(KK+II-1,2)-YS1
           SAVE(KK+II-1,1)=SAVE(KK+II-1,1)-XS2
892       CONTINUE
*
          IF (MFLAG.LE.2)
     1       CALL MINCH(CRMAX,KZT,SAVE(KK,1),SAVE(KK,2),
     2       CRTAB(1,1,2),ITGT(L,2),ISAV(KK),APPRCH,
     3       CRIT(L,2),NFILL,CUTS,ARFILL)
*
          IF (MFLAG.GE.3)
     1       CALL MINCH(CRMAX,KZT,SAVE(KK,2),SAVE(KK,1),
     2       CRTAB(1,1,2),ITGT(L,2),ISAV(KK),APPRCH,
     3       CRIT(L,2),NFILL,CUTS,ARFILL)
*
          DO 893 II=1,KZT
           SAVE(KK+II-1,2)=SAVE(KK+II-1,2)+YS1
           SAVE(KK+II-1,1)=SAVE(KK+II-1,1)+XS2
893       CONTINUE
          ARFILS=ARFILS+ARFILL
        ENDIF
630     NUMAPR(MFLAG,2)=KZT
        KZ=KZ1+NFILL
        KZ1=KZ
        NUMAPR(MFLAG,1)=NFILL
        FILL=FILL+FLOAT(NFILL)
        IF (KZ.EQ.KN1) GO TO 670
        GO TO (640,650,660,670),MFLAG
*
        PRINT*,'ERR GOTO ORIGINAL LINE NUMBER 733'
*
640     MFLAG=2
        NFILL=0
        KP1=KP1+KZT
        XS1=TGT(L,4)-CRIT(L,2)
        IF (XC(L)+CRIT(L,2).LE.TGT(L,4)) THEN
          XS2=XC(L)
          GO TO 360
        ENDIF
```

```
*
650       MFLAG=3
          KP1=KP1-NUMAPR(1,2)+NUMAPR(1,1)+NFILL
          CALL SORT(K-KP1,SAVE(KP1,1),SAVE(KP1,2),
     1              SAVE(KP1,3),ISAV(KP1))
          XS1=CRIT(L,2)
          YS1=0.
          XS2=0.
          YS2=YC(L)+CRIT(L,2)
          GO TO 560
*
660       MFLAG=4
          KP1=KP1+KZT
          XS1=TGT(L,4)
          XS2=TGT(L,4)-CRIT(L,2)
          GO TO 560
*
670       KZ=KH(L)
          IF ((IREPR.GE.10).AND.(FILL.GT.0.)) THEN
            WRITE(13,890)L,KH(L),K0,FILL,
     1           (SAVE(KK,1),SAVE(KK,2),SAVE(KK,3),KK=1,M)
            KZ=KH(L)+IFIX(FILL+.01)
            IF (L.GT.1) THEN
              K0=IFIX(FILL+.01)
              DO 690 KZ1=1,K0
               KK=K0+KZ1-1
               S1=SAVE(KK,1)
               S2=SAVE(KK,2)
               S3=SAVE(KK,3)
               IS=ISAV(KK)
               KZP=KH(L)+KZ1+1
               DO 680 KM=KZP,KK
                KM1=KK-KM+KZP
                SAVE(KM1,1)=SAVE(KM1-1,1)
                SAVE(KM1,2)=SAVE(KM1-1,2)
                SAVE(KM1,3)=SAVE(KM1-1,3)
                ISAV(KM1)=ISAV(KM1-1)
680            CONTINUE
               KZP=KZP-1
               SAVE(KZP,1)=S1
               SAVE(KZP,2)=S2
               SAVE(KZP,3)=S3
               ISAV(KZP)=IS
690           CONTINUE
            ENDIF
          ENDIF
700       SIGCRT(L)=SIGCRT(L)+FLOAT(NMIN)**2
          ENAPFL(L)=ENAPFL(L)+FILL
          SNAPFL(L)=SNAPFL(L)+FILL**2
          APRMIN(L)=FILL
          GO TO 720
        ENDIF
720     ADM(L)=ADM(L)+SUNRUN
```

```
         ITGTGP=ITGT(L,3)
         GPADAC(ITGTGP)=GPADAC(ITGTGP)+SUMRUN
         SIGADM(L)=SIGADM(L)+SUMRUN*SUMRUN
         RAPF(L)=RAPF(L)+ARFILS
         SIGNAF(L)=SIGNAF(L)+ARFILS*ARFILS
         IF (CRIT(L,1).GT.0.) APRA(L)=ARFILS
       ENDIF
730    CONTINUE
       L=L+1
       K0=K
       FILL=0.
       ARFILL=0.
       ARFILS=0.
       CUTS=0.
       SUMRUN=0.
       IF (SAVE(K,3).GT.FLOAT(L)) GO TO 430
       IF ((K.EQ.M).AND.(SAVE(K,3).EQ.FLOAT(L))) GO TO 430
       IF ((L.LE.NELT).AND.(K.EQ.M)) GO TO 430
      ENDIF
740   CONTINUE
      DO 750 J=1,NTGPS
       GPADMS(J)=GPADMS(J)+GPADAC(J)**2
750   CONTINUE
      II3=1
*
*------COMPUTE COMBINED PROBABILITIES FOR RUNWAY, TAXIWAY,AND SOD
*
      IF (NCP.GT.1) THEN
        I3=0
        KJ=1
        IFIN=0
        DO 790 JJ=1,2
         DO 790 JK=1,NCP
*
*----------ONLY INTERESTED IN 1&2 (KJ=1), 1&3 (KJ=2), 2&3 (KJ=3)
*
          IF (JJ.GE.JK) GO TO 790
          IF (IPCUT(II3).EQ.0) GO TO 760
          IF (IPCUT(JJ).NE.1) II3=JJ
          IF (IPCUT(JK).NE.1) II3=JK
760       IF ((IPCUT(JJ).NE.1).OR.(IPCUT(JK).NE.1)) GO TO 780
*
*----------BOTH SURFACES ARE CUT
*
          I3=I3+1
*
*-----II INDICATES WHICH SURFACE HAS THE MINIMUM NUMBER OF CRATERS TO
*     REPAIR FOR COMBINATIONS OF 2 SURFACES AND II3 FOR ALL 3 SURFACES
*
          II=JJ
          IF (IHIT(JJ).GT.IHIT(JK)) II=JK
          IF (IHIT(II3).GT.IHIT(JK)) II3=JK
```

```
*
*----------DISTRIBUTION OF MINIMUM NUMBER OF CRATERS
*
770       ICUT(KJ,II)=ICUT(KJ,II)+1
          I2CUT(KJ)=I2CUT(KJ)+1
          SGCRAT(KJ)=SGCRAT(KJ)+FLOAT(IHIT(II))**2
*
*----------MINIMUM NUMBER OF CRATERS
*
          ICRAT(KJ)=ICRAT(KJ)+IHIT(II)
*
*----------AREA OF CRATERS
*
          SMINA(KJ)=SMINA(KJ)+AMIN(II)
          SGMINA(KJ)=SGMINA(KJ)+AMIN(II)**2
*
*----------MINIMUM NUMBER OF CRATERS ON APPROACH TO OPERATIONAL STRIP
*
          SAPR(KJ)=SAPR(KJ)+APRMIN(II)
          SGAPR(KJ)=SGAPR(KJ)+APRMIN(II)**2
*
*----------AREA OF CRATERS ON APPROACH
*
          SAPRA(KJ)=SAPRA(KJ)+APRA(II)
          SGAPRA(KJ)=SGAPRA(KJ)+APRA(II)**2
          IF (IFIN.EQ.1) GO TO 800
780       KJ=KJ+1
          IF ((JJ.NE.2).OR.(JK.NE.3)) GO TO 790
*
*-----ALL COMBINATIONS OF 2 SURFACES HAVE BEEN LOOKED AT. IF ALL 3
*     SURFACES HAVE BEEN CUT (I3=3) COMPUTE STATISTICS FOR ALL 3 & EXIT
*     LOOP (IFIN=1).
*
          IF (I3.NE.3) GO TO 800
          KJ=4
          II=II3
          IFIN=1
          GO TO 770
790     CONTINUE
       ENDIF
800    CALL REPAIR(MXPTCH,KZ,M0,IREPR,CRMAX,II3,NAREA,NCP)
       M=M0
       M0=0
       KZ=0
       II=0
810    IF (IT.GT.1) THEN
         IF (MOD(IT,NSAMPT).EQ.0) CALL RESLTS
       ENDIF
820   CONTINUE
```

```
*
*-----TEST TO SEE IF LIMITING MONTE CARLO LOOP WAS DESIRED
*     AND APPROPRIATE. IF NOT, AVOID SUBROUTINE "NCOMP".
*
      IF ((NFLAG3.EQ.1).AND.(NSAMP.GE.200)) THEN
*
*-----TESTS ON FLAGS SET INSIDE SUBROUTINE "NCOMP" TO DIRECT
*     EITHER RETURN TO MONTE CARLO LOOP OR PASS ON, BASED ON
*     ESTIMATE OF ITERATIONS REQUIRED.
*
        IF (NFLAG2.EQ.0) CALL NCOMP
825     IF (NFLAG1.EQ.0) THEN
          NFLAG1=1
          GO TO 85
        ENDIF
      ENDIF
*
*-----CALCULATE AND PRINT STATISTICS
*
830   IF (MOD((IT-1),NSAMPT).NE.0) CALL RESLTS
      CLOSE(UNIT=13)
*
840   FORMAT (1X,'NO HITS DURING ATTACK, MONTE CARLO ITERATION: ',I4)
890   FORMAT (8H TARGET ,I3,9H KH(L) = ,I4,6H K0 = ,I4,8H FILL = ,F7.0,8
     100(/1X,3F10.2))
901   FORMAT(A6)
905   FORMAT('1       INPUT FILE: ',A6,'     OUTPUT FILE: ',A6,//)
991   FORMAT(I10)
1200  FORMAT (1H0,37HMORE THAN 800 HITS WERE FOUND IN PASS,I4,1H./1X,20H
     1EXCESS WERE IGNORED.)
1934  FORMAT(A1)
1935  FORMAT(A6)
1950  FORMAT(4I10)
1955  FORMAT(5(F15.4,1X),3I10)
1960  FORMAT(4I6)
1961  FORMAT(10(F9.3,1X))
1965  FORMAT(6F12.1)
1970  FORMAT('1',T20,'*** DATA INPUT ECHO ***',//)
1975  FORMAT(2I10,F10.1,I10)
1980  FORMAT(I10,2F10.4)
      END
**********************************************************************
* LAST UPDATE 23/2300 JAN 84          FILE:SUBS1.AAP
**********************************************************************
      SUBROUTINE TRISUB(RV)
      U=RANF()
      X=SQRT(2.0*U)
      RV=1000.0*X-1000.0
      RETURN
      END
```

```
*-----------------------------------------------------------------------
      SUBROUTINE NORAN(R,SR,D,SD)
*
      COMMON/RAY/TWOPI
*
      X=RANF()
      A=SQRT(-2.*ALOG(X))
      X=RANF()
      X=TWOPI*X
      R=A*SR*SIN(X)
      D=A*SD*COS(X)
      RETURN
      END
*-----------------------------------------------------------------------
      SUBROUTINE INITL(NELT,NTGPS,NCP,LV)
*
      COMMON
     1 ADM(112)       ,GPHT(15)       ,MXPTCH         ,SIGADM(112),
     2 AMIN(3)        ;GPHTAC(15)                     ,SIGARP(3),
     3 APRA(3)        ,GPHTS(15)                      ,SIGASP(3),
     4 APRMIN(3)      ,LNHITS(112)    ,NSAMP1         ,SIGCRT(3),
     5 AREP(3)        ,ICRAT(4)       ,PASS(0:32,6)   ,SIGCTS(27),
     6 ASTP(3)        ,ICUT(4,3)      ,PATT(13,34)    ,SIGFIL(27),
     7 COUNTR(112)    ,IHIT(3)        ,RAPF(112)      ,SIGHTS(112),
     8 CRIT(112,2)    ,IPASS(32,2)    ,RCUT(112)      ,SIGNAF(112),
     9 CRTAB(11,6,2)  ,IPAT(12,4)     ,RHIT(112)      ,SMINA(4),
     & DECAR(112)     ,IPCUT(3)       ,SAPR(4)        ,SNAPFL(3),
     1 DSTR(3)                        ,SAPRA(4)       ,TGT(112,5),
     2 ENAPFL(3)      ,IPL(40)        ,SAVE(800,3)    ,XC(3),
     3 GPADAC(15)     ,ISAV(800)      ,SGAPR(4)       ,YC(3),
     4 GPADM(15)      ,ITGT(112,3)    ,SGAPRA(4),
     5 GPADNS(15)     ,I2CUT(4)       ,SGCRAT(4),
     6 GPAREA(15)     ,KH(3)          ,SGMINA(4)
*
      DO 10 I=1,NELT
       COUNTR(I)=0.
       SIGHTS(I)=0.
       ADM(I)=0.
       SIGADM(I)=0.
10    CONTINUE
      DO 20 J=1,NTGPS
       GPHTS(J)=0.
       GPADNS(J)=0.
20    CONTINUE
      IPAV=LV+NCP
      DO 30 K=1,IPAV
       RAPF(K)=0.
       SIGNAF(K)=0.
       RCUT(K)=0.
       RHIT(K)=0.
       SIGCTS(K)=0.
       SIGFIL(K)=0.
30    CONTINUE
```

```
      DO 40 L=1,NCP
       SIGCRT(L)=0.
       ASTP(L)=0.
       SIGASP(L)=0.
       AREP(L)=0.
       ENAPFL(L)=0.
       SNAPFL(L)=0.
       IHIT(L)=0
       IPCUT(L)=0
       AMIN(L)=0.
       APRMIN(L)=0.
       APRA(L)=0.
       DSTR(L)=0.
       SIGARP(L)=0.
40    CONTINUE
      N1=NCP+1
      DO 50 I=1,N1
       I2CUT(I)=0
       ICRAT(I)=0
       SGCRAT(I)=0.
       SMINA(I)=0.
       SGMINA(I)=0.
       SAPR(I)=0.
       SGAPR(I)=0.
       SAPRA(I)=0.
       SGAPRA(I)=0.
       DO 50 J=1,NCP
        ICUT(I,J)=0
50    CONTINUE
      RETURN
      END
X-----------------------------------------------------------------------
      SUBROUTINE SORT(N,XI,YI,ZI,IX)
X
      DIMENSION IX(N),ZI(N),XI(N),YI(N)
X
      EQUIVALENCE (IT,T)
X
      JO=0
10    JO=JO+JO+1
      IF (JO.LT.N) GO TO 10
20    JO=JO/2
      IF (JO.LE.0) RETURN
      KO=N-JO
      DO 40 LO=1,KO
       MO=LO
30     NO=MO+JO
       IF (XI(MO).GT.XI(NO)) THEN
         T=XI(MO)
         XI(MO)=XI(NO)
         XI(NO)=T
         T=YI(MO)
         YI(MO)=YI(NO)
```

```
      YI(NO)=T
      T=ZI(MO)
      ZI(MO)=ZI(NO)
      ZI(NO)=T
      IT=IX(MO)
      IX(MO)=IX(NO)
      IX(NO)=IT
      MO=MO-JO
      IF (MO.GT.0) GO TO 30
     ENDIF
40   CONTINUE
     GO TO 20
     END
X-----------------------------------------------------------------------
     SUBROUTINE BLDG(XI,YI,CRTAB,L,NP,N,TL,TW,AREA)
X
     DIMENSION XI(N),YI(N),CRTAB(11,6,2),NP(N)
X
X----ASSESS AREA REMAINING UNDAMAGED AFTER ALL HITS ARE
X    EVALUATED FOR THIS ATTACK
X
     RATIO=TL/TW
     DO 10 J=1,N
      DW=SQRT(AREA/RATIO)
      DL=DW*RATIO
      XH=.5*(TL-DL)
      YH=.5*(TW-DW)
      XOC=.5*TL-XH
      YOC=.5*TW-YH
      XCEN=XI(J)-XH
      YCEN=YI(J)-YH
      D1=ABS(YCEN-YOC)
      D2=ABS(XCEN-XOC)
      NPJ=NP(J)
      IF ((D1.LT.(CRTAB(L,NPJ,1)+0.5*DW)).AND.
    1    (D2.LT.(CRTAB(L,NPJ,1)+0.5*DL))) THEN
       KA=1
       IF ((D1.LE.(0.5*TW)).AND.(D2.LE.(0.5*TL))) KA=2
       DWDTH=AMIN1(DW,YCEN+CRTAB(L,NPJ,KA))
       DWDTH=DWDTH-AMAX1(0.,YCEN-CRTAB(L,NPJ,KA))
       DLNGTH=AMIN1(DL,XCEN+CRTAB(L,NPJ,KA))
       DLNGTH=DLNGTH-AMAX1(0.,XCEN-CRTAB(L,NPJ,KA))
       DAREA=DLNGTH*DWDTH
       AREA=AREA-DAREA
       IF (AREA.LE.0.) RETURN
      ENDIF
10   CONTINUE
     RETURN
     END
X-----------------------------------------------------------------------
```

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X LAST UPDATE 16/2300 JAN 84            FILE:SUBS2.AAP
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
      SUBROUTINE CLSTRP(CRMAX,N,XI,YI,CRTAB,LT,NP,TL,TW,CL,CW,XSTAR,
     1                   YSTAR,ICSTAR)
      DIMENSION XI(N),YI(N),CRTAB(11,6),AREA(800),ISORT(800),JSORT(800),
     1         NP(N)
      XC=0.0
      YC=0.0
      TSXU=CL
      TSYU=CW
      CSTAR=10.0E15
      ICSTAR=N
X
X-----DEFINE AREA(J)=DIFFICULTY OF REPAIRING CRATER J
X     CHANGED 28 OCT 81 TO COMPUTE AREA OF SQUARE CRATERS
X
      DO 24 J=1,N
       AREA(J)=4.0XCRTAB(LT,NP(J))XX2
24    CONTINUE
X
X-----SET UP FOR SWEEP
X
25    NMIN=0
      ISTART=0
      SWEP=10.E15
      DO 11 J=1,N
       IF ((YI(J)+CRTAB(LT,NP(J)).GT.YC).AND.
     1      (YI(J)-CRTAB(LT,NP(J)).LT.TSYU)) THEN
X
14        IF (NMIN.EQ.0) THEN
            NMIN=1
            ISORT(1)=J
            JSORT(1)=J
            GO TO 11
          ENDIF
X
5         IT=NMIN
          NMIN=NMIN+1
17        JZ=ISORT(IT)
X
          IF ((XI(J)+CRTAB(LT,NP(J))).LT.(XI(JZ)+CRTAB(LT,NP(JZ)))) THEN
            ISORT(IT+1)=ISORT(IT)
            IT=IT-1
            IF (IT.GT.0) GO TO 17
            ISORT(1)=J
          ELSE
18          ISORT(IT+1)=J
          ENDIF
X
116       IT=NMIN-1
117       JR=JSORT(IT)
X
```

```
        IF ((YI(J)+CRTAB(LT,NP(J))).LT.(YI(JR)+CRTAB(LT,NP(JR)))) THEN
          JSORT(IT+1)=JSORT(IT)
          IT=IT-1
          IF (IT.GT.0) GO TO 117
          JSORT(1)=J
        ELSE
118       JSORT(IT+1)=J
        ENDIF
      ENDIF
11    CONTINUE
*
*-----EXECUTE SWEEP
*     DETERMINE DIFFICULTY OF REPAIRING CRATERS TOUCHING FRAME
*
10    IX=ISTART+1
      AICC=0.0
      ICC=0
30    IF (IX.LE.NMIN) THEN
        JN=ISORT(IX)
        IF ((XI(JN)-CRTAB(LT,NP(JN))).LT.TSXU) THEN
          AICC=AICC+AREA(JN)
          ICC=ICC+1
31        IX=IX+1
          GO TO 30
        ELSE
32        IF ((XI(JN)-CRMAX).LT.TSXU) THEN
            IX=IX+1
            GO TO 30
          ENDIF
        ENDIF
      ENDIF
*
*-----COMPARE REPAIR DIFFICULTY FOR FRAME
*
6     IF (CSTAR.GT.AICC) THEN
        CSTAR=AICC
        ICSTAR=ICC
        XSTAR=XC
        YSTAR=YC
        IF (CSTAR.LE.0.0000001) THEN
          XSTAR=XSTAR+CL
          RETURN
        ENDIF
      ENDIF
*
*-------MOVE FRAME
*
16    TEMP=AICC-CSTAR
41    ISTART=ISTART+1
```

```
      IF (ISTART.LE.NMIN) THEN
        IS=ISORT(ISTART)
        IF (TEMP.GT.AREA(IS)) THEN
          TEMP=TEMP-AREA(IS)
          GO TO 41
        ENDIF
998     IF (SWEP.GT.AICC) SWEP=AICC
        TSXU=XI(IS)+CRTAB(LT,NP(IS))+CL+0.000000001
        IF (TSXU.LE.TL) THEN
          XC=TSXU-CL
          GO TO 10
        ENDIF
      ENDIF
X
X-----SWEEP FINISHED
X
20    TEMP=SWEP-CSTAR
      JDP=0
46    JDP=JDP+1
      IF (JDP.GT.NMIN) THEN
        XSTAR=XSTAR+CL
        RETURN
      ENDIF
      IS=JSORT(JDP)
      IF (TEMP.GT.AREA(IS)) THEN
        TEMP=TEMP-AREA(IS)
        GO TO 46
      ENDIF
45    TSYU=YI(IS)+CRTAB(LT,NP(IS))+CW+0.000000001
      IF (TSYU.GT.TW) THEN
        XSTAR=XSTAR+CL
        RETURN
      ENDIF
      YC=TSYU-CW
      XC=0.0
      TSXU=CL
      GO TO 25
      END
X-------------------------------------------------------------------
      SUBROUTINE MINCW(CRMAX,N,X,Y,CR,LT,KP,W,WW,NREP,CUTS,ATOTAL)
X
X-----HARNETT'S TAXIWAY PROGRAM INSERTED TO REPLACE MINCW 1 OCT 81
X     LATEST VERSION OF TAXIWAY  23 APRIL 1982
X           NC = MAX NUMBER OF CRATERS IN A SUBPROBLEM
X           NSUB = MAX NUMBER OF SUBPROBLEMS TO BE SOLVED
X           N = NUMBER OF CRATERS IN ENTIRE PROBLEM
X
       DIMENSION ISTART(1001),A(100),X(N),Y(N),CR(11,6),
     1          LIST1(50),LIST2(50),IT(50),WX(50),WY(50),WR(50),
     2          IREP(50),KP(N),IPSOL(50),ICOMP(50),IBEAS(50)
X
      COMMON/TAXI/NFN,NF,NL
X
```

```
      CRMAX=0.0
      IF (N.GT.50) THEN
        WRITE(6,799)N
        CALL EXIT
      ENDIF
*
*------CHANGED TO COMPUTE AREA OF SQUARE CRATERS 28 OCT 81
*
750   DO 100 J=1,N
       IF (CRMAX.LT.CR(LT,KP(J))) CRMAX=CR(LT,KP(J))
       A(J)=4.0*CR(LT,KP(J))**2
100   CONTINUE
*
      NREP=0
      ATOTAL=0.0
*
*-----SEARCH FOR SUBPROBLEMS
*
      ISTART(1)=1
      NSUB=1
      NNM=N-1
      DO 110 J=1,NNM
       JP=J+1
       JM=J
       EL=X(J)+CR(LT,KP(J))
       EU=X(JP)-CR(LT,KP(JP))
       IF ((EL+W).LE.EU) THEN
*
101      JM=JM-1
         IF (JM.GE.1) THEN
           IF ((X(JM)+CR(LT,KP(JM))).GT.EL) EL=X(JM)+CR(LT,KP(JM))
           IF ((X(JM)+CRMAX).GT.EL) GO TO 101
         ENDIF
*
103      JP=JP+1
         IF (JP.LE.N) THEN
           IF (EU.GT.(X(JP)-CR(LT,KP(JP)))) EU=X(JP)-CR(LT,KP(JP))
           IF (EU.GT.(X(JP)-CRMAX)) GO TO 103
         ENDIF
*
105      IF ((EL+W).LE.EU) THEN
           NSUB=NSUB+1
           IF (NSUB.GT.1000) THEN
             WRITE(6,798)
             CALL EXIT
           ENDIF
760        ISTART(NSUB)=J+1
         ENDIF
       ENDIF
110   CONTINUE
      ISTART(NSUB+1)=N+1
```

```
*
*-----SOLVE SUBPROBLEMS
*
      DO 230 JS=1,NSUB
      NF=ISTART(JS)
      NL=ISTART(JS+1)-1
      NFM=NF-1
      CRMAX=0.0
      DO 5 J=NF,NL
       IF (CRMAX.LT.CR(LT,KP(J))) CRMAX=CR(LT,KP(J))
5     CONTINUE
      NC=NL-NFM
      IF (NC.GT.50) THEN
        WRITE(6,797)NC
        CALL EXIT
      ENDIF
770   IF (NC.LE.2) THEN
        BFEAS=0.0
        NP=NF+1
        IF (Y(NF)+CR(LT,KP(NF)).GT.WW-W) THEN
          IF (Y(NF)-CR(LT,KP(NF)).GE.W) GO TO 122
          BFEAS=BFEAS+A(NF)
          NREP=NREP+1
          IREP(NREP)=NF
          ATOTAL=ATOTAL+A(NF)
          IF (NC.LE.1) GO TO 230
          IF (Y(NP)+CR(LT,KP(NP)).LE.WW-W) GO TO 230
          IF (Y(NP)-CR(LT,KP(NP)).GE.W) GO TO 230
          BFEAS=BFEAS+A(NP)
          NREP=NREP+1
          IREP(NREP)=NP
          ATOTAL=ATOTAL+A(NP)
          GO TO 230
        ENDIF
112     IF (NC.LE.1) GO TO 230
        IF (Y(NP)+CR(LT,KP(NP)).LE.WW-W) GO TO 230
        IF (Y(NP)-CR(LT,KP(NP)).GE.W) GO TO 114
113     ATOTAL=ATOTAL+A(NP)
        BFEAS=BFEAS+A(NP)
        NREP=NREP+1
        IREP(NREP)=NP
        GO TO 230
114     XD=X(NF)-X(NP)
        YD=Y(NF)-Y(NP)
        DIST=SQRT(XD**2+YD**2)-2.0*CR(LT,KP(NP))
        IF (DIST.GE.W) GO TO 230
        IF (((Y(NF)-CR(LT,KP(NF))).GE.W).AND.
     1      ((Y(NP)-CR(LT,KP(NP))).GE.W)) GO TO 230
        AMIN=A(NF)
        ISAVE=NF
        IF (A(NF).GT.A(NP)) ISAVE=NP
        IF (A(NF).GT.A(NP)) AMIN=A(NP)
        ATOTAL=ATOTAL+AMIN
```

```
          NREP=NREP+1
          IREP(NREP)=ISAVE
          BFEAS=BFEAS+AMIN
          GO TO 230
122       IF (NC.LE.1) GO TO 230
          IF (Y(NP)-CR(LT,KP(NP)).GE.W) GO TO 230
          IF (Y(NP)+CR(LT,KP(NP)).LE.(WW-W)) GO TO 114
          GO TO 113
        ENDIF
X
X------CHECK CLEAR PATH
X
1       DO 2 J = 1,NC
         IPSOL(J)=0
2       CONTINUE
        CALL CHECK(IPSOL,IFLAG,X,Y,CR,WX,WY,WR,NC,LIST1,LIST2,IT,
     1            LT,KP,CRMAX,WW,W)
        IF (IFLAG.LE.0) GO TO 6000
        BFEAS=0.0
        GO TO 200
X
X------INITIALIZATION FOR IMPLICIT ENUMERATION
X
6000    DO 7500 K=1,NC
         IBEAS(K)=0
         ICOMP(K)=1
7500    CONTINUE
X
        JLAST=0
        ITER=0
        NREPC=0
        REP=0
        BFEAS=10.E20
X
X------FORWARD MOVE
X
7000    JLAST=JLAST+1
        IUNDER=JLAST
        IPSOL(JLAST)=1
        REP=REP+A(NFM+JLAST)
X
X------TEST 2
X
        IF (REP.GE.BFEAS) GO TO 7020
X
X------TEST 1
X
        CALL CHECK(IPSOL,IFLAG,X,Y,CR,WX,WY,WR,NC,LIST1,LIST2,IT,
     1            LT,KP,CRMAX,WW,W)
        IF (IFLAG.LE.0) GO TO 7010
        BFEAS=REP
```

```
      DO 7030 K=1,NC
       IBEAS(K) = IPSOL(K)
7030  CONTINUE
*
*------TEST 6
*
7020  IF (NREPC.EQ.JLAST) GO TO 70
*
*------BACKWARD MOVE
*
      NREPC=NREPC+IUNDER-JLAST+1
      IPSOL(IUNDER)=0
      JLAST=IUNDER
      REP=REP-A(NFM+JLAST)
      IF (JLAST.LE.1) GO TO 7010
      M=IUNDER-1
      DO 7040 K=1,M
       L=IUNDER-K
       IF (IPSOL(L).EQ.1) THEN
         IUNDER=IUNDER-K
         GO TO 7010
       ENDIF
7040  CONTINUE
7010  IF (JLAST.EQ.NC) GO TO 7050
      M=JLAST+1
      RMIN=10000.0
      DO 7060 K=M,NC
       IF (A(NFM+K).LT.RMIN) RMIN=A(NFM+K)
7060  CONTINUE
7050  BND=REP+RMIN
*
*------TEST 3
*
      IF ((BND.GE.BFEAS).OR.(JLAST.EQ.NC)) GO TO 7020
*
*------TEST 4
*
      IF (IPSOL(JLAST).EQ.1) GO TO 7001
      DO 7070 K=1,JLAST
       ICOMP(K)=IPSOL(K)
7070  CONTINUE
      CALL CHECK(ICOMP,IFLAG,X,Y,CR,WX,WY,WR,NC,LIST1,LIST2,IT,
     1           LT,KP,CRMAX,WM,W)
*
*------TEST 5
*
      IF (IUNDER.NE.JLAST) THEN
        M=IUNDER+1
        DO 7080 K=M,NC
         ICOMP(K) = 1
7080    CONTINUE
      ENDIF
7001  IF (IFLAG.LE.0) GO TO 7020
```

```
      GO TO 7000
70    ATOTAL=ATOTAL+BFEAS
200   CONTINUE
      IF (BFEAS.GT.0.0) THEN
        DO 201 I=1,NC
         IF (IBEAS(I).GT.0) THEN
           NREP=NREP+1
           IREP(NREP)=NFM+I
         ENDIF
201     CONTINUE
      ENDIF
230   CONTINUE
      CUTS=0.
      IF (NREP.NE.0) CUTS=FLOAT(NSUB)
      RETURN
797   FORMAT(1H0,10X,49HNUMBER OF CRATERS IN SUBPROGRAM EXCEEDS 50, NC=
     1 ,I5)
798   FORMAT(1H0,10X,23HSUBPROBLEMS EXCEED 1000)
799   FORMAT (1H0,10X,33HNUMBER OF CRATERS EXCEEDS  50, N= ,I5)
      END
*-----------------------------------------------------------------------
************************************************************************
* LAST UPDATE 16/2300 JAN 84            FILE:SUBS3.AAP
************************************************************************
      SUBROUTINE CHECK(IN,IFLAG,X,Y,CR,WX,WY,WR,NC,LIST1,LIST2,IT,
     1                 LT,KP,CRMAX,WW,W)
*
      DIMENSION IN(NC),IT(NC),WX(NC),WY(NC),WR(NC),X(NC),Y(NC),
     1          CR(11,6),LIST1(NC),LIST2(NC),KP(NC)
*
      COMMON/TAXI/NFM,NF,NC
*
      IFLAG=1
      JT=0
      DO 6 JX=1,NC
       IF (IN(JX).LT.1) THEN
         JT=JT+1
         JJ=NFM+JX
         WX(JT)=X(JJ)
         WY(JT)=Y(JJ)
         WR(JT)=CR(LT,KP(JJ))
       ENDIF
6     CONTINUE
      IF (JT.LE.0) RETURN
      IT(1)=-1
      IF ((WY(1)-WR(1)).GE.W) IT(1)=0
      IF ((WY(1)+WR(1)).LE.(WW-W)) IT(1)=1
      IF (IT(1).LT.0) THEN
        IFLAG=0
        RETURN
      ENDIF
      JX=1
10    JX=JX+1
```

```
      JXM=JX-1
      IF (JX.GT.JT) RETURN
*
*-----CAN WE GET OVER JX?
*
      IF ((WY(JX)+WR(JX)).LE.(WW-W)) THEN
        IF (IT(JXM).GT.0) THEN
*
*---------DO AN 'OVER-OVER'
*
          XMIN=WX(JX)-WR(JX)-CRMAX-W
*
*---------CHECK BACK
*
          JTEMP=JXM
13        JTEMP=JTEMP-1
          IF (JTEMP.GT.0) THEN
*
*-----------DOES AN 'UNDER' IMPINGE UPON JX?
*
            XD=WX(JX)-WX(JTEMP)
            YD=WY(JX)-WY(JTEMP)
            DIF=SQRT(XD**2+YD**2)-WR(JX)-WR(JTEMP)
            IF (DIF.LT.W) THEN
              IF (IT(JTEMP).LE.0) GO TO 12
            ENDIF
15          IF (WX(JTEMP).GE.XMIN) GO TO 13
          ENDIF
16        IT(JX)=1
          GO TO 10
        ENDIF
*
*-------TRY FOR 'UNDER-OVER'
*
11      JFLAG=1
        CALL BETWN(JXM,JX,JT,JFLAG,WX,WY,WR,LIST1,LIST2,CRMAX,WW,W)
        IF (JFLAG.GT.0) THEN
          IT(JX)=1
          GO TO 10
        ENDIF
      ENDIF
*
*-----CAN WE GET UNDER JX ?
*
12    IF ((WY(JX)-WR(JX)).LT.W) THEN
        JFLAG=0
        RETURN
      ENDIF
20    IF (IT(JXM).LT.1) THEN
*
*-------DO AN 'UNDER-UNDER'
*
        XMIN=WX(JX)-WR(JX)-CRMAX-W
```

```
*
*-------DOES AN 'OVER' IMPINGE UPON JX ?
*
        JTEMP=JXM
17      JTEMP=JTEMP-1
        IF (JTEMP.GT.0) THEN
          XD=WX(JX)-WX(JTEMP)
          YD=WY(JX)-WY(JTEMP)
          DIF=SQRT(XD**2+YD**2)-WR(JX)-WR(JTEMP)
          IF (DIF.LT.W) THEN
            IF (IT(JTEMP).GE.1) GO TO 500
          ENDIF
18        IF (WX(JTEMP).GE.XMIN) GO TO 17
        ENDIF
19      IT(JX)=0
        GO TO 10
      ENDIF
*
*-----TRY FOR 'OVER-UNDER'
*
14    JFLAG=2
      CALL BETWN(JXM,JX,JT,JFLAG,WX,WY,WR,LIST1,LIST2,CRMAX,WW,W)
      IF (JFLAG.GT.0) THEN
        IT(JX)=0
        GO TO 10
      ENDIF
*
*-----BACKTRACK
*
500   JI=JX
501   JI=JI-1
      IF (JI.GE.1) THEN
        IF (IT(JI).LE.0) GO TO 501
        JX=JI
        JXM=JX-1
        IF (JXM.GT.0) THEN
          IF ((WY(JX)-WR(JX)).GE.W) GO TO 20
          GO TO 500
        ENDIF
502     IF ((WY(1)-WR(1)).GE.W) THEN
          IT(1)=0
          GO TO 10
        ENDIF
      ENDIF
999   IFLAG=0
      RETURN
      END
```

```
*-----------------------------------------------------------------
      SUBROUTINE BETWN(JXM,JX,JT,JFLAG,WX,WY,WR,LIST1,LIST2,CRMAX,WW,W)
*
      DIMENSION WX(JT),WY(JT),WR(JT),LIST1(JT),LIST2(JT)
*
      COMMON /TAXI/NFM,NF,NL
*
*-----(JFLAG.LE.1) IMPLIES 'UNDER-OVER'
*     (JFLAG.GE.2) IMPLIES 'OVER-UNDER'
*
      KFLAG=1
      NL1=1
      LIST1(1)=JX
      NLT=1
      K=JX
      XMIN=WX(JX)-WR(JX)-CRMAX-W
*
*-----CONSTRUCT 'LIST1' OF CRATERS BEHIND JX IMPINGING
*     DIRECTLY OR INDIRECTLY UPON IT
*
1     KM=JXM
*
*-----DETERMINE IF KM IMPINGES UPON K
*
2     IF (WX(KM).GE.XMIN) THEN
        DO 13 IX=1,NL1
         IF (KM.EQ.LIST1(IX)) GO TO 3
13      CONTINUE
        XD=WX(K)-WX(KM)
        YD=WY(K)-WY(KM)
        DIS=SQRT(XD**2+YD**2)-WR(KM)-WR(K)
        IF (DIS.LT.W) THEN
          IF ((JFLAG.LE.1).AND.((WY(KM)+WR(KM)).GT.(WW-W))) GO TO 999
          IF ((JFLAG.GE.2).AND.((WY(KM)-WR(KM)).LT.W)) GO TO 999
*
*---------DETERMINE IF KM IMPINGES UPON JXM
*
          XD=WX(KM)-WX(JXM)
          YD=WY(KM)-WY(JXM)
          DIS=SQRT(XD**2+YD**2)-WR(KM)-WR(JXM)
          IF (DIS.LT.W) GO TO 999
          TEMP=WX(KM)-WR(KM)-CRMAX-W
          IF (XMIN.GT.TEMP) XMIN=TEMP
          NL1=NL1+1
          LIST1(NL1)=KM
        ENDIF
3       KM=KM-1
        IF (KM.GT.0) GO TO 2
      ENDIF
4     NLT=NLT+1
```

```
      IF (NLT.LE.NL1) THEN
        K=LIST1(NLT)
        GO TO 1
      ENDIF
*
*-----CONSTRUCT 'LIST2' OF CRATERS AHEAD OF JXM IMPINGING
*     DIRECTLY OR INDIRECTLY UPON IT
*
5     NL2=1
      LIST2(1)=JXM
      NLT=1
      K=JXM
      XMAX=WX(K)+WR(K)+CRMAX+W
*
*-----DETERMINE IF KP IMPINGES UPON K
*
7     KP=JX
8     IF (WX(KP).LE.XMAX) THEN
        DO 19 IX=1,NL2
          IF (KP.EQ.LIST2(IX)) GO TO 9
19      CONTINUE
        XD=WX(K)-WX(KP)
        YD=WY(K)-WY(KP)
        DIS=SQRT(XD**2+YD**2)-WR(KP)-WR(K)
        IF (DIS.LT.W) THEN
          IF ((JFLAG.LE.1).AND.((WY(KP)-WR(KP)).LT.W)) GO TO 999
          IF ((JFLAG.GE.2).AND.((WY(KP)+WR(KP)).GT.(WW-W))) GO TO 999
*
*---------DETERMINE IF KP IMPINGES UPON JX
*
          XD=WX(KP)-WX(JX)
          YD=WY(KP)-WY(JX)
          DIS=SQRT(XD**2+YD**2)-WR(KP)-WR(JX)
          IF (DIS.LT.W) GO TO 999
          TEMP=WX(KP)+WR(KP)+CRMAX+W
          IF (XMAX.LT.TEMP) XMAX=TEMP
          NL2=NL2+1
          LIST2(NL2)=KP
        ENDIF
9       KP=KP+1
        IF (KP.LE.JT) GO TO 8
      ENDIF
10    NLT=NLT+1
      IF (NLT.LE.NL2) THEN
        K=LIST2(NLT)
        GO TO 7
      ENDIF
*
*-----DETERMINE IF LIST1 IMPINGES UPON LIST2
*
```

```
1000  DO 30 K1=1,NL1
       L1=LIST1(K1)
       DO 30 K2=1,NL2
        L2=LIST2(K2)
        DX=WX(L1)-WX(L2)
        DY=WY(L1)-WY(L2)
        DIS=SQRT(DX**2+DY**2)-WR(L1)-WR(L2)
        IF (DIS.LT.W) GO TO 999
30    CONTINUE
      GO TO 2000
999   KFLAG=0
2000  JFLAG=KFLAG
      RETURN
      END
*---------------------------------------------------------------------
      SUBROUTINE OVLAP(X,Y,CRTAB,LT,NP,X0,Y0,ITL,ITW,KZ,SUM)
*
      COMMON/RAY2/SQUARE(900),CRMAX
*
      DIMENSION X(KZ),Y(KZ),CRTAB(11,6),NP(KZ)
*
*-----INITIALIZE
*
      DO 10 I=1,ITW
       SQUARE(I)=0.
10    CONTINUE
      SUM=0.
      SUMP=0.
*
*-----FIND FIRST AND LAST VALUES OF X TO CONSIDER
*
      L3=MAX1(1.,(X(1)-CRMAX+1.-X0))
      L2=MIN1(FLOAT(ITL),(X(KZ)+CRMAX+1.-X0))
      J6=1
      M=0
20    L1=L3
*
*-----LOOP-ONE SQUARE AT A TIME IN X
*     L=X VALUE AT TOP OF SQUARE
*
      DO 120 L=L1,L2
       DXP=0.
       J6=J6+M
       M=0
       IF(J6.GT.KZ) RETURN
*
*------IF ALL CRATERS HAVE BEEN CONSIDERED, RETURN
*      LOOP-CRATER BY CRATER...CONSIDER ALL CRATERS WHICH
*      COULD POSSIBLY INTERSECT IN X
*
       DO 90 I=J6,KZ
```

```
*
*-------LOCATE LEFT HAND EDGE OF CRATER
*
      NPI=NP(I)
      X1=X(I)-CRTAB(LT,NPI)-X0
      IF (X1.LT.FLOAT(L-1)) GO TO 30
      X2=X(I)-CRMAX-X0
      IF (X2.GE.FLOAT(L)) GO TO 100
      IF (X1.GE.FLOAT(L)) GO TO 90
*
*-------LEFT-HAND EDGE OF CRATER LIES INSIDE LTH SQUARE
*
      DXP=FLOAT(L)-X1
      GO TO 60
*
*-------LEFT HAND EDGE OF CRATER IS BELOW X-SQUARE
*       LOCATE RIGHT HAND EDGE OF CRATER
*
30    X1=X(I)+CRTAB(LT,NPI)-X0
      IF (X1.LE.FLOAT(L-1)) GO TO 40
      IF (X1.GE.FLOAT(L)) GO TO 50
*
*-------RIGHT HAND EDGE OF CRATER LIES INSIDE LTH SQUARE
*
      DXP=X1-FLOAT(L)+1.
      GO TO 60
*
*-------CRATER I LIES ENTIRELY LEFT OF X-SQUARE...NO NEED TO CONSIDER
*       THIS CRATER ANY MORE
*
40    X3=X(I)+CRMAX-X0
      IF (X3.LE.FLOAT(L-1)) N=N+1
      GO TO 90
50    DXP=1.
*
*-------CRATER INTERSECTS X-SQUARE...CHECK INTERSECTIONS IN Y
*
60    Y1=Y(I)-CRTAB(LT,NPI)-Y0
*
*-------K1=INDEX OF Y-SQUARE CONTAINING LOWER EDGE OF CRATER I
*
      K1=MAX1(1.,Y1+1.)
*
*-------D1=% OF Y-SQUARE OCCUPIED BY CRATER
*
      D1=AMIN1(1.,FLOAT(K1)-Y1)
      SQUARE(K1)=D1*DXP+SQUARE(K1)
      IF (K1.EQ.ITW) GO TO 90
      K1=K1+1
      Y1=Y(I)+CRTAB(LT,NPI)-Y0
      K2=MIN0(ITW,IFIX(Y1))
      IF (K2.EQ.ITW) GO TO 70
      D1=Y1-FLOAT(K2)
```

```
*
*-------LOAD SQUARE CONTAINING TOP EDGE OF CRATER I
*
        SQUARE(K2+1)=D1*DXP+SQUARE(K2+1)
*
*-------LOAD INTERMEDIATE Y-SQUARES...D1=1.
*
70      DO 80 J=K1,K2
         SQUARE(J)=SQUARE(J)+DXP
80      CONTINUE
90     CONTINUE
*
*-------COUNT SQUARES THAT ARE AT LEAST HALF-FILLED
*
100    DO 110 J=1,ITW
        IF (SQUARE(J).GE.0.5) SUMP=SUMP+1.
        SQUARE(J)=0.
110    CONTINUE
       SUM=SUM+SUMP
*
*------IF THERE IS A GAP IN X-VALUES, SKIP TO NEXT X-VALUE NEEDED
*
       IF (DXP.LE.0.) THEN
         IF (M.NE.0) THEN
           J6PM=J6+M
           IF (J6PM.GT.KZ) RETURN
           L3=IFIX(X(J6PM)-CRMAX-X0)+1
           IF (L3.GT.L) GO TO 20
           L3=L+1
           GO TO 20
         ENDIF
       ENDIF
       SUMP=0.
120   CONTINUE
      RETURN
      END
*-----------------------------------------------------------------------
```

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X LAST UPDATE 14/2200 JAN 84            FILE:SUBS4.AAP
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
      SUBROUTINE REPAIR(MXP,KZ,M0,IREPR,CRMAX,II3,NAREA,NCP)
X
      COMMON
     1 ADM(112)       ,GPHT(15)       ,MXPTCH        ,SIGADM(112),
     2 AMIN(3)        ,GPHTAC(15)                    ,SIGARP(3),
     3 APRA(3)        ,GPHTS(15)                     ,SIGASP(3),
     4 APRMIN(3)      ,LNHITS(112)    ,NSAMP1        ,SIGCRT(3),
     5 AREP(3)        ,ICRAT(4)       ,PASS(0:32,6)  ,SIGCTS(27),
     6 ASTP(3)        ,ICUT(4,3)      ,PATT(13,34)   ,SIGFIL(27),
     7 COUNTR(112)    ,IHIT(3)        ,RAPF(112)     ,SIGHTS(112),
     8 CRIT(112,2)    ,IPASS(32,2)    ,RCUT(112)     ,SIGNAF(112),
     9 CRTAB(11,6,2)  ,IPAT(12,4)     ,RHIT(112)     ,SMINA(4),
     & DECAR(112)     ,IPCUT(3)       ,SAPR(4)       ,SNAPFL(3),
     1 DSTR(3)                        ,SAPRA(4)      ,TGT(112,5),
     2 ENAPFL(3)      ,IPL(40)        ,SAVE(800,3)   ,XC(3),
     3 GPADAC(15)     ,ISAV(800)      ,SGAPR(4)      ,YC(3),
     4 GPADM(15)      ,ITGT(112,3)    ,SGAPRA(4),
     5 GPADMS(15)     ,I2CUT(4)       ,SGCRAT(4),
     6 GPAREA(15)     ,KH(3)          ,SGMINA(4)
X
      NREP=MIN0(KZ,MXP)
      IF (NREP.EQ.0) RETURN
      K1=0
      K9=KZ
      KTYP=MOD(IREPR,10)
      IF (KTYP.GT.0) THEN
        IF ((SAVE(1,3).LT.FLOAT(II3)).OR.(KTYP.EQ.2)) THEN
          IF ((SAVE(1,3).GT.FLOAT(II3)).AND.(KTYP.EQ.2)) RETURN
          DO 10 J=1,KZ
           IF (SAVE(J,3).GT.FLOAT(II3)) GO TO 20
           IF (SAVE(J,3).LT.FLOAT(II3)) K1=J
10        CONTINUE
20        K9=J-1
        ENDIF
      ENDIF
30    K9=MIN0(K9,NREP+K1)
      K1=K1+1
      IF (K9.LT.K1) THEN
        IF (KTYP.EQ.2) RETURN
        K1=0
        K9=KZ
        GO TO 30
      ENDIF
40    L=IFIX(SAVE(K1,3)+.01)
      SUMR=KH(L)-K1+1
      IF (NAREA.EQ.0) SUMR=AMIN(L)
```

```
      IF (K9.LT.KH(L)) THEN
        SUMR=K9-K1+1
        IF (NAREA.EQ.0) THEN
          IF (SUMR.LE.FLOAT(KH(L)-K9)) THEN
            SUMR=0.
            CALL OVLAP(SAVE(K1,1),SAVE(K1,2),CRTAB,ITGT(L,2),ISAV(K1),
     1                 XC(L)-CRIT(L,1),YC(L),IFIX(CRIT(L,1)),
     2                 IFIX(CRIT(L,2)),K9-K1+1,SUMR)
            GO TO 60
          ENDIF
50        J=K9+1
          SUMR=0.
          CALL OVLAP (SAVE(J,1),SAVE(J,2),CRTAB,ITGT(L,2),ISAV(J),
     1                 XC(L)-CRIT(L,1)-2.*CRMAX,YC(L)-2.*CRMAX,
     2                 IFIX(CRIT(L,1)+4.*CRMAX),IFIX(CRIT(L,2)+4.*CRMAX),
     3                 KH(L)-K9,SUMR)
          SUMR=AMIN(L)-SUMR
        ENDIF
      ENDIF
60    AREP(L)=AREP(L)+SUMR
      SIGARP(L)=SIGARP(L)+SUMR**2
      K5=MIN0(K9,KH(L))+1
      DO 70 J=K5,N0
       J1=K1+J-K5
       SAVE(J1,1)=SAVE(J,1)
       SAVE(J1,2)=SAVE(J,2)
       SAVE(J1,3)=SAVE(J,3)
       ISAV(J1)=ISAV(J)
70    CONTINUE
      K5=K5-K1
      NREP=NREP-K5
      MXP=MXP-K5
      K2=K2-K5
      N0=N0-K5
      DO 80 J=L,NCP
       KH(J)=KH(J)-K5
80    CONTINUE
      IF ((NREP.EQ.0).OR.(K2.EQ.0)) RETURN
      IF (SAVE(K1,3).NE.FLOAT(L)) THEN
        IF (KTYP.EQ.2) RETURN
        K1=0
        K9=K2
        GO TO 30
      ENDIF
*
*-----REPAIR HITS ON APPROACH FOR LTH TARGET -- IF APPROPRIATE
*
90    DO 100 J=K1,K2
       IF (SAVE(J,3).NE.FLOAT(L)) GO TO 110
       IF(J-K1+1.GT.NREP) GO TO 110
       ITGTTP=ITGT(L,2)
       JMPNTP=ISAV(J)
       IF (NAREA.EQ.0) SUMR=SUMR+4.*CRTAB(ITGTTP,JMPNTP,1)**2
```

```
100   CONTINUE
110   K5=J-K1
      WRITE(13,150)K1,K2,M0,J,(SAVE(KK,1),SAVE(KK,2),SAVE(KK,3),KK=1,M0)
      DO 120 J1=J,M0
       KK=K1+J1-J
       SAVE(KK,1)=SAVE(J1,1)
       SAVE(KK,2)=SAVE(J1,2)
       SAVE(KK,3)=SAVE(J1,3)
       ISAV(KK)=ISAV(J1)
120   CONTINUE
      IF (NAREA.EQ.1) SUMR=K5
      WRITE(13,160)K5
      NREP=NREP-K5
      MXP=MXP-K5
      K2=K2-K5
      M0=M0-K5
      L=L+1
      IF (L.LE.NCP) THEN
        DO 130 J=L,NCP
         KH(J)=KH(J)-K5
130     CONTINUE
      ENDIF
140   IF ((NREP.EQ.0).OR.(K2.EQ.0)) RETURN
      IF (KTYP.EQ.2) RETURN
      K1=0
      K9=K2
      GO TO 30
150   FORMAT (6H K1 = ,I3,6H K2 = ,I3,6H M0 = ,I4,5H J = ,I4,800(/1X,3F1
     12.2))
160   FORMAT (40H NUMBER OF CRATERS FILLED ON APPROACH = ,I6)
      END
```

```
*-----------------------------------------------------------------------------
      SUBROUTINE RESLTS
*
      CHARACTER NAME*4
*
      DIMENSION PR1(15),PR2(15),PR3(15),PR4(15),PP5(15),PR6(15)
*
      COMMON
     1 ADM(112)      ,GPHT(15)      ,MXPTCH        ,SIGADM(112),
     2 AMIN(3)       ,GPHTAC(15)                   ,SIGARP(3),
     3 APRA(3)       ,GPHTS(15)                    ,SIGASP(3),
     4 APRMIN(3)     ,LNHITS(112)   ,NSAMP1        ,SIGCRT(3),
     5 AREP(3)       ,ICRAT(4)      ,PASS(0:32,6)  ,SIGCTS(27),
     6 ASTP(3)       ,ICUT(4,3)     ,PATT(13,34)   ,SIGFIL(27),
     7 COUNTR(112)   ,IHIT(3)       ,RAPF(112)     ,SIGHTS(112),
     8 CRIT(112,2)   ,IPASS(32,2)   ,RCUT(112)     ,SIGNAF(112),
     9 CRTAB(11,6,2) ,IPAT(12,4)    ,RHIT(112)     ,SMINA(4),
     & DECAR(112)    ,IPCUT(3)      ,SAPR(4)       ,SNAPFL(3),
     1 DSTR(3)                      ,SAPRA(4)      ,TGT(112,5),
     2 ENAPFL(3)     ,IPL(40)       ,SAVE(800,3)   ,XC(3),
     3 GPADAC(15)    ,ISAV(800)     ,SGAPR(4)      ,YC(3),
     4 GPADM(15)     ,ITGT(112,3)   ,SGAPRA(4),
     5 GPADMS(15)    ,I2CUT(4)      ,SGCRAT(4),
     6 GPAREA(15)    ,KH(3)         ,SGMINA(4)
*
      COMMON/END/NSAMP,NELT,NTGPS,NCP,CRMIN,APPRCH,NAREA
*
      COMMON/JOHN/NFLAG1,NFLAG2,NMAX,NSAMPR,ZALPH,ERROR,NSAMP2,NFLAG3
*
      NAME=' NO'
      SAMPL=1./FLOAT(NSAMP)
      SAMPO=FLOAT(NSAMP-1)
      DO 10 I=1,NTGPS
       GPAREA(I)=0.
       GPADM(I)=0.
       GPHT(I)=0.
10    CONTINUE
      CT=0.
      DO 30 L=1,NELT
       IF (COUNTR(L).GT.CT) THEN
         LCOUNT=L
         CT=COUNTR(L)
       ENDIF
       ITGTGP=ITGT(L,3)
       GPHT(ITGTGP)=GPHT(ITGTGP)+COUNTR(L)
       GPADM(ITGTGP)=GPADM(ITGTGP)+ADM(L)
       GPAREA(ITGTGP)=GPAREA(ITGTGP)+TGT(L,4)*TGT(L,5)
30    CONTINUE
      CONF90=SIGHTS(LCOUNT)-SAMPL*COUNTR(LCOUNT)**2
      CONF90=SQRT(CONF90/SAMPO)
      CONF90=2.576*CONF90*SQRT(SAMPL)
      WRITE(13,240)NSAMP,CONF90,LCOUNT
      CONF90=1.645*CONF90/2.576
```

```
      WRITE(13,250)CONF90
      IF (NFLAG3.EQ.1.AND.NSAMP.GE.200) WRITE(13,490)
      IB=0
40    IA=IB+1
      IB=MIN0(IA+14,NELT)
      KM=IB-IA+1
      WRITE(13,260)(K,K=IA,IB)
      DO 50 K=1,KM
       L=K+IA-1
       PR1(K)=SAMPL*COUNTR(L)
       PR2(K)=SIGHTS(L)-SAMPL*COUNTR(L)**2
       PR2(K)=SQRT(PR2(K)/SAMPO)
       PR3(K)=SAMPL*ADM(L)
       PR4(K)=SIGADM(L)-SAMPL*ADM(L)**2
       PR4(K)=SQRT(PR4(K)/SAMPO)
50    CONTINUE
      WRITE(13,270)(PR1(K),K=1,KM)
      WRITE(13,280)(PR2(K),K=1,KM)
      IF (NAREA.EQ.0) WRITE(13,290)(PR3(K),K=1,KM)
      IF (NAREA.EQ.0) WRITE(13,300)(PR4(K),K=1,KM)
      WRITE(13,310)(ITGT(K,3),K=IA,IB)
      IF (IB.LT.NELT) GO TO 40
      WRITE(13,320)
      IB=0
60    IA=IB+1
      IB=MIN0(IA+14,NTGPS)
      KM=IB-IA+1
      WRITE(13,330)(K,K=IA,IB)
      DO 70 K=1,KM
       L=K+IA-1
       PR1(K)=GPHTS(L)-SAMPL*GPHT(L)**2
       PR1(K)=SQRT(PR1(K)/SAMPO)
       GPHT(L)=SAMPL*GPHT(L)
       PR2(K)=GPADMS(L)-SAMPL*GPADM(L)**2
       PR2(K)=SQRT(PR2(K)/SAMPO)
       GPADM(L)=SAMPL*GPADM(L)
       GPAREA(L)=GPADM(L)/GPAREA(L)
70    CONTINUE
      WRITE(13,270)(GPHT(K),K=IA,IB)
      WRITE(13,280)(PR1(K),K=1,KM)
      IF (NAREA.EQ.0) THEN
        WRITE(13,290)(GPADM(K),K=IA,IB)
        WRITE(13,300)(PR2(K),K=1,KM)
        WRITE(13,340)(GPAREA(K),K=IA,IB)
      ENDIF
80    IF (IB.LT.NTGPS) GO TO 60
```

```
      IF (NCP.GT.0) THEN
       WRITE(13,350)NAME
       DO 120 L=1,NCP
        PR1(1)=SAMPL*RCUT(L)
        PR1(2)=SQRT((PR1(1)-PR1(1)**2)*SAMPL)
        PR1(4)=SIGCRT(L)-SAMPL*RHIT(L)**2
        PR1(4)=SQRT(PR1(4)/SAMP0)
        PR1(3)=SAMPL*RHIT(L)
        PR1(5)=SAMPL*ASTP(L)
        PR1(6)=SIGASP(L)-SAMPL*ASTP(L)**2
        PR1(6)=SQRT(PR1(6)/SAMP0)
        PR1(7)=SAMPL*AREP(L)
        PR1(8)=SIGARP(L)-SAMPL*AREP(L)**2
        PR1(8)=SQRT(PR1(8)/SAMP0)
        PR1(12)=SIGNAF(L)-SAMPL*RAPF(L)**2
        PR1(12)=SQRT(PR1(12)/SAMP0)
        PR1(11)=SAMPL*RAPF(L)
        PR1(10)=SNAPFL(L)-SAMPL*ENAPFL(L)**2
        PR1(10)=SQRT(PR1(10)/SAMP0)
        PR1(9)=SAMPL*ENAPFL(L)
        IF (NAREA.EQ.1) THEN
          PR1(5)=1.E20
          PR1(6)=1.E20
        ENDIF
90      IF (MXPTCH.EQ.0) THEN
          PR1(7)=1.E20
          PR1(8)=1.E20
        ENDIF
100     IF (APPROW.LT.1.) THEN
          PR1(9)=1.E20
          PR1(10)=1.E20
          PR1(11)=1.E20
          PR1(12)=1.E20
        ENDIF
110     WRITE(13,360)L,CRIT(L,1),CRIT(L,2),(PR1(K),K=1,12)
120    CONTINUE
       IF (NCP.GT.1) THEN
        WRITE(13,370)
        IEL1=1
        IEL2=2
        NCP1=NCP+1
        DO 170 KJ=1,NCP1
         KK=4-KJ
         DO 130 L=1,3
          DSTR(L)=SAMPL*FLOAT(ICUT(KJ,L))
          IF (KK.GT.0) DSTR(KK)=1.E20
130      CONTINUE
         PR1(1)=SAMPL*FLOAT(I2CUT(KJ))
         PR1(2)=SQRT(SAMPL*(PR1(1)-PR1(1)**2))
         PR1(4)=FLOAT(ICRAT(KJ))
         PR1(3)=SAMPL*PR1(4)
         PR1(4)=SGCRAT(KJ)-SAMPL*PR1(4)**2
         PR1(4)=SQRT(PR1(4)/SAMP0)
```

```
      PR1(5)=SAMPL*SMINA(KJ)
      PR1(6)=SSMINA(KJ)-SAMPL*SMINA(KJ)**2
      PR1(6)=SQRT(PR1(6)/SAMPO)
      PR1(7)=SAMPL*SAPR(KJ)
      PR1(8)=SGAPR(KJ)-SAMPL*SAPR(KJ)**2
      PR1(8)=SQRT(PR1(8)/SAMPO)
      PR1(9)=SAMPL*SAPRA(KJ)
      PR1(10)=SGAPRA(KJ)-SAMPL*SAPRA(KJ)**2
      PR1(10)=SQRT(PR1(10)/SAMPO)
      IF (NAREA.EQ.1) THEN
        PR1(5)=1.E20
        PR1(6)=1.E20
      ENDIF
140   IF (APPRCH.LT.1.) THEN
        PR1(7)=1.E20
        PR1(8)=1.E20
        PR1(9)=1.E20
        PR1(10)=1.E20
      ENDIF
150   IF (KJ.LT.4) THEN
        IF (KJ.EQ.2) IEL2=3
        IF (KJ.EQ.3) IEL1=2
        WRITE(13,400)IEL1,IEL2,CRIT(IEL1,1),CRIT(IEL2,2),(PR1(K),
     1               K=1,6),(DSTR(K),K=1,3),(PR1(K),K=7,10)
        IF (NCP.EQ.3) GO TO 170
        GO TO 180
      ENDIF
160   WRITE(13,380)CRIT(1,1),CRIT(1,2),(PR1(K),K=1,6),(DSTR(K),
     1              K=1,3),(PR1(K),K=7,10)
170     CONTINUE
       ENDIF
      ENDIF
180   IF (LV.GT.0) WRITE(13,390)
      LV=0
      DO 190 L=1,NELT
       IF ((ITST(L,1).EQ.1).AND.(CRIT(L,1).LT.1.)) THEN
        LV=LV+1
        IPL(LV)=L
       ENDIF
190   CONTINUE
      IF (LV.GT.0) THEN
        IB=0
200     IA=IB+1
        IB=MIN0(IA+14,LV)
        KN=IB-IA+1
        WRITE(13,400)(IPL(K),K=IA,IB)
*-------NON-ANSI STANDARD SUBSCRIPTS MAY REQUIRE ADJUSTMENT.
        WRITE(13,410)(TGT(IPL(K),5),K=IA,IB)
        WRITE(13,420)(CRIT(IPL(K),2),K=IA,IB)
```

```
      DO 210 K=1,KM
       L=K+IA-1
       IPLL=IPL(L)
       PR1(K)=SAMPL*RCUT(IPLL)
       PR2(K)=SIGCTS(L)-SAMPL*RCUT(IPLL)**2
       PR2(K)=SQRT(PR2(K)/SAMPO)
       PR3(K)=SAMPL*RHIT(IPLL)
       PR4(K)=SIGFIL(L)-SAMPL*RHIT(IPLL)**2
       PR4(K)=SQRT(PR4(K)/SAMPO)
       PR6(K)=SAMPL*RAPF(IPLL)
       PR5(K)=SIGNAF(IPLL)-SAMPL*RAPF(IPLL)**2
       PR5(K)=SQRT(PR5(K)/SAMPO)
210   CONTINUE
      WRITE(13,430)(PR1(K),K=1,KM)
      WRITE(13,440)(PR2(K),K=1,KM)
      WRITE(13,450)(PR3(K),K=1,KM)
      WRITE(13,440)(PR4(K),K=1,KM)
      IF (NAREA.EQ.0) THEN
        WRITE(13,460)(PR6(K),K=1,KM)
        WRITE(13,470)(PR5(K),K=1,KM)
      ENDIF
220   IF (IB.LT.LV) GO TO 200
     ENDIF
     RETURN
240  FORMAT(1X,'NSAMP =',I5,5X,'CONF INTERVAL FOR 99% LEVEL =',F7.3,
    12X,'FOR TGT ELT =',I5)
250  FORMAT(10X,29HCONF INTERVAL FOR 90% LEVEL =,F7.3)
260  FORMAT(1H0,1X,11HTGT ELEMENT,15I8)
270  FORMAT(1X,12HEXP NO. HITS,15F8.3)
280  FORMAT(8X,5HSIGMA,15F8.3)
290  FORMAT(1X,12HEXP AREA DAM,15F8.0)
300  FORMAT(8X,5HSIGMA,15F8.0)
310  FORMAT(2X,11HTGT GP. NO.,15I8)
320  FORMAT(1H0,13HTARGET GROUPS)
330  FORMAT(1H0,1X,11HTGT GP. NO.,15I8)
340  FORMAT(1X,12HEXP PER. DAM,15F8.3)
350  FORMAT (1H0,4X,30HFOR RUNWAYS AND MAJOR TAXIWAYS,/8X,3HTGT,4X,3HMC
    1L,2X,3HMCW,3X,4HPROB,2X,5HSIGMA,2X,6HEXP NO,3X,5HSIGMA,3X,8HEXP AR
    2EA,3X,5HSIGMA,3X,4HEXP ,A4,3X,5HSIGMA,3X,8HEXP APPR,3X,5HSIGMA,3X,
    38HEXP APPR,3X,5HSIGMA,/8X,3HELT,16X,3HCUT,8X,7HCRATERS,15X,4HFILL,
    415X,6HFILLED,12X,7HNO CRAT,15X,4HFILL)
360  FORMAT(8X,I3,F7.0,F5.0,2F7.3,2F8.3,4X,F7.0,1X,F7.0,4X,F7.0,1X,F7.
    10,3X,F8.3,1X,F8.3,3X,F8.0,1X,F8.0)
370  FORMAT (1H0,4X,29HCOMBINED PROBABILITIES OF CUT,/77X,12HDISTRIBUTI
    1ON,/75X,16HMINIMUM  CRATERS,/8X,3HTGT,4X,3HMCL,2X,3HMCW,3X,4HPROB,
    22X,5HSIGMA,2X,6HEXP NO,3X,5HSIGMA,3X,8HEXP AREA,3X,5HSIGMA,4X,3(3H
    3ELT,3X),8HEXP APPR,3X,5HSIGMA,3X,8HEXP APPR,3X,5HSIGMA,/7X,4HELTS,
    416X,3HCUT,8X,7HCRATERS,15X,4HFILL,13X,1H1,5X,1H2,5X,1H3,5X,7HNO CR
    5AT,15X,4HFILL)
380  FORMAT(6X,5H1&2&3,F7.0,F5.0,2F7.3,2F8.3,4X,F7.0,1X,F7.0,3X,3(F5.3
    1,1X),1X,2F8.3,3X,2F8.0)
390  FORMAT(1H0,4X,18HFOR MINOR TAXIWAYS)
400  FORMAT(1H0,13X,14HTARGET ELEMENT,15I7)
```

```
410   FORMAT(16X,12HTARGET WIDTH,15F7.0)
420   FORMAT(9X,19HMINIMUM CLEAR WIDTH,15F7.0)
430   FORMAT(5X,23HEXPECTED NUMBER OF CUTS,15F7.3)
440   FORMAT(23X,5HSIGMA,15F7.3)
450   FORMAT(4X,24HEXPECTED CRATERS TO FILL,15F7.3)
460   FORMAT(7X,21HEXPECTED AREA TO FILL,15F7.0)
470   FORMAT(23X,5HSIGMA,15F7.0)
480   FORMAT(8X,I1,1H&,I1,F7.0,F5.0,2F7.3,2F8.3,4X,F7.0,1X,F7.0,3X,3(F5.
     13,1X),1X,2F8.3,3X,2F8.0)
490   FORMAT(1H ,'NSAMP LIMITED TO LEAST OF VALUE INPUT OR NUMBER NEEDED
     = TO GIVE SPECIFIED QUALITY TO PROBABILITY OF CUT.')
      END
```

```
X-----------------------------------------------------------------
      SUBROUTINE NCOMP
X
X-----THIS ROUTINE IS ENTERED TO CALCULATE THE MINIMUM SAMPLE SIZE
X     OF MONTE CARLO ITERATIONS TO GIVE A SPECIFIC CONFIDENCE LEVEL
X     AND INTERVAL FOR THE PROBABILITY OF CUTTING A TAKEOFF
X     SURFACE. IT CANNOT BE ENTERED UNLESS NFLAG3 IS SET IN MAIN
X     PROGRAM AND NSAMP SPECIFIED AS GREATER THAN 200.
X
      COMMON
     1 ADM(112)       ,GPHT(15)       ,MXPTCH         ,SIGADM(112),
     2 AMIN(3)        ,GPHTAC(15)                     ,SIGARP(3),
     3 APRA(3)        ,GPHTS(15)                      ,SIGASP(3),
     4 APRMIN(3)      ,LNHITS(112)    ,NSAMP1         ,SIGCRT(3),
     5 AREP(3)        ,ICRAT(4)       ,PASS(8:32,6)   ,SIGCTS(27),
     6 ASTP(3)        ,ICUT(4,3)      ,PATT(13,34)    ,SIGFIL(27),
     7 COUNTR(112)    ,IHIT(3)        ,RAPF(112)      ,SIGHTS(112),
     8 CRIT(112,2)    ,IPASS(32,2)    ,RCUT(112)      ,SIGNAF(112),
     9 CRTAB(11,6,2)  ,IPAT(12,4)     ,RHIT(112)      ,SMINA(4),
     & DECAR(112)     ,IPCUT(3)       ,SAPR(4)        ,SNAPFL(3),
     1 DSTR(3)                        ,SAPRA(4)       ,TGT(112,5),
     2 ENAPFL(3)      ,IPL(40)        ,SAVE(800,3)    ,XC(3),
     3 GPADAC(15)     ,ISAV(800)      ,SGAPR(4)       ,YC(3),
     4 GPADM(15)      ,ITGT(112,3)    ,SGAPRA(4),
     5 GPADMS(15)     ,I2CUT(4)       ,SGCRAT(4),
     6 GPAREA(15)     ,KH(3)          ,SGMINA(4)
X
      COMMON/END/NSAMP2,NELT,NTGPS,NCP,CRMIN,APPRCH,NAREA
X
      COMMON/JOHN/NFLAG1,NFLAG2,NMAX,NSAMPR,ZALPH,ERROR,NSAMP,NFLAG3
X
      DIMENSION PR(3)
X
X-----CALCULATE AND STORE IN A MATRIX THE PROBABILITY OF CUT FOR
X     EACH TARGET ELEMENT, USING THIS PATTERN.
X
      IF (NCP.GE.1) THEN
        IF (ZALPH.LT.1.645) ZALPH=1.645
        IF ((ERROR.GT.0.05).OR.(ERROR.LT.0.0001)) ERROR=0.05
        DO 10 J=1,NCP
         PR(J)=RCUT(J)/FLOAT(NSAMP)
10      CONTINUE
X
X-------INITIALIZE A LOOP TO FIND THAT PROBABILITY OF CUT CLOSEST
X       TO 0.5. THIS MAXIMIZES REQUIRED SAMPLE SIZE FOR WORST CASE
X       TARGET ELEMENT AND ATTACK.
X
        SMALL=ABS(PR(1)-0.5)
        IX=1
        JX=1
```

```
*
*-------LOOP TO FIND PROBABILITY OF CUT CLOSEST TO 0.5
*       AND RECORD IT AS PKNUM.
*
        DO 20 J=1,NCP
         SMALL1=ABS(PR(J)-0.5)
         IF (SMALL1.LT.SMALL) THEN
           IX=I
           JX=J
           SMALL=SMALL1
         ENDIF
20      CONTINUE
        PKNUM=PR(JX)
        NUM=0
*
*-------IF PKNUM IS VERY CLOSE TO ZERO OR ONE, THE STATISTICS
*       COLLAPSE MONTE CARLO ITERATIONS TO A VERY SMALL NUMBER.
*       THEN CALCULATION OF ADDITIONAL ITERATIONS TO RUN OR
*       RETURN TO THE MONTE CARLO LOOP SHOULD NOT BE COMPLETED.
*       THIS ACCOMPLISHED BY SETTING NFLAG1.
*
        IF ((PKNUM.GT.0.0009).AND.(PKNUM.LT.0.9995)) THEN
*
*---------CALCULATE TOTAL SAMPLE SIZE TO ASSURE CONFIDENCE LEVEL
*         AND ERROR INTERVAL.
*
          SSIZE=PKNUM*(1.-PKNUM)*((ZALPH/ERROR)**2.)
          NUM=SSIZE+1.
*
*---------TEST IF MORE ITERATIONS REQUIRED, SETTING APPROPRIATE FLAGS
*         WHETHER TO RETURN TO THE MONTE CARLO LOOP. IF SO, SET LOWER
*         AND UPPER MONTE CARLO LOOP LIMITS.
*
          IF (NUM.LE.NSAMP) THEN
            NFLAG1=1
            RETURN
          ELSE
            NSAMPR=NSAMP+1
            NFLAG2=1
            IF (NUM.LT.NMAX) THEN
             NSAMP=NUM
            ELSE
              NSAMP=NMAX
            ENDIF
            RETURN
          ENDIF
        ENDIF
      ENDIF
95    NFLAG1=1
      RETURN
      END
```

APPENDIX D

SAMPLE DATABASES AND INPUT FILE

Sample Target Database

2[a] 1[b] 0[c] 0[d]

4500.0[e] 0.0[f] 0.0[g] 9000.0[h] 200.0[i] 1[j] 1[k] 1[l]

4000.0[m] 50.0[n]

1500.0 2598.0 60.0 5999.9 150.0 1 2[o] 1

4000.0 50.0

2[p] 0[q]

2[r] 2[s]

0[t] 1[u]

	Variable
a - number of target elements	NELT
b - number of target groups	NTGPS
c - minimum width required to taxi aircraft (0=no taxi search)	NPPRCW
d - flag to compute overlapping craters (0 = compute, 1 = do not)	NAREA
e - target element 1 center-point x-coordinate	TGT(1,1)
f - target element 1 center-point y-coordinate	TGT(1,2)
g - target element 1 azimuth of primary axis (orientation)	TGT(1,3)
h - target element 1 length	TGT(1,4)
i - target element 1 width	TGT(1,5)
j - surface code (0 = non-surface, 1 = surface)	ITGT(1,1)
k - target element 1 hardness code (crater table index)	ITGT(1,2)
l - target group of target element 1	ITGT(1,3)
m - minimum clear length (MCL) for target element 1	CRIT(1,1)
n - minimum clear width (MCW) for target element 1	CRIT(1,2)
o - target element 2 hardness code (crater table index)	ITGT(2,2)
p - number of TOL surfaces	NCP
q - number of taxi surfaces	LV
r - total number of hardness codes	NHARD
s - number hardness codes for surfaces	NHARDP
t - number of patches available	MXPTCH
u - repair priority code	IREPR

Sample Weapon/Attack Pattern Database

2[a] 2[b] 2[c]

25.5[d] 15.6[e]

12.2[f] 6.7[g]

26.6[h] 17.7[i]

13.3[j] 8.9[k]

1[l]

6[m] 1[n] 1[o] 0[p]

300.0[q] 150.0[r] 30.0[s] 30.0[t] 0.0[u] 0.0[v] 0.0[w] 0.0[x] 0.990[y] 0.000[z]

-150.0[A]	10.0[B]
-100.0	-10.0
-50.0	10.0
50.0	-10.0
100.0	10.0
150.0	-10.0

	Variable
a - total number of hardness levels	NHARD
b - number of hardness levels for pavements	NHARDP
c - number of weapons defined	NWPN
d - hardness 1, weapon 1 deny-TOL crater size	CRTAB(1,1,1)
e - hardness 1, weapon 2 deny-TOL crater size	CRTAB(1,2,1)
f - hardness 1, weapon 1 deny-taxi crater size	CRTAB(1,1,2)
g - hardness 1, weapon 2 deny-taxi crater size	CRTAB(1,2,2)
h - hardness 2, weapon 1 deny-TOL crater size	CRTAB(2,1,1)
i - hardness 2, weapon 2 deny-TOL crater size	CRTAB(2,2,1)
j - hardness 2, weapon 1 deny-taxi crater size	CRTAB(2,1,2)
k - hardness 2, weapon 2 deny-taxi crater size	CRTAB(2,2,2)
l - number of attack patterns defined	NPATT
m - number of weapons in pattern	IPAT(1,1)
n - number of submunitions in weapon	IPAT(1,2)
o - weapon code (crater table index number identifying weapon)	IPAT(1,3)
p - pattern shape (0=gen purpose,1=rec CBU,2=elip CBU,3=guided)	IPAT(1,4)
q - range delivery error standard deviation	PATT(1,1)
r - deflection delivery error standard deviation	PATT(1,2)
s - range ballistic error standard deviation	PATT(1,3)
t - deflection ballistic error standard deviation	PATT(1,4)
u - CBU half-pattern length	PATT(1,5)
v - CBU half-pattern width	PATT(1,6)
w - CBU half-void length	PATT(1,7)
x - CBU half-void width	PATT(1,8)
y - munition fuze reliability	PATT(1,9)
z - submunition fuze reliability	PATT(1,10)
A - munition range coordinate relative to center of stick	PATT(1,11)
B - munition deflection coordinate relative to center stick	PATT(1,12)

Sample AAPMSN Output File/AAPMOD Input File

```
512597745[a]
    200[b]     200[c]
      0[d]   .0500[e]    1.645[f]

    2[g]          1[h]       0[i]         0[j]
 4500.0[k]      0.0[l]     0.0[m]    9000.0[n]    200.0[o]    1[p]    1[q]    1[r]
 4000.0[s]     50.0[t]
 1500.0      2598.0       60.0       5999.9       150.0       1       2       1
 4000.0        50.0

 2[u]   0[v]

 1[w]
 6[x]   1[y]   1[z]   0[A]

 300.0[B]  150.0[C]  30.0[D]  30.0[E]  0.0[F]  0.0[G]  0.0[H]  0.0[I]  0.990[J]  0.000[K]

 -150.0[L]   10.0[M]
 -100.0     -10.0
  -50.0      10.0
   50.0     -10.0
  100.0      10.0
  150.0     -10.0

 2[N]   2[O]

 25.5[P]   15.6[Q]
 12.2       6.7
 26.6      17.7
 13.3       8.9

 0[R]   1[S]   2[T]

 3000.0[U]   0.0[V]   170.0[W]   .950[X]   1.0[Y]   1[Z]   0[aa]   1[bb]

 1500.0   2598.0   135.0   .960   1.0   1   0   2
```

	Variable
a - random number seed (if required)	ISEED
b - total number of samples to be run	NSAMP
c - interval between reports	NSAMPT
d - flag to calculate number of required samples (0 = perform NSAMP samples, 1 = as calculate > 200)	NFLAG3
e - required level of significance (error allowable)	ERROR
f - standard normal variate associated with allowable error	ZALPHA
g - number of target elements	NELT
h - number of target groups	NTGPS
i - minimum width required to taxi aircraft (0=no taxi search)	NPPRCW
j - flag to compute overlapping craters (0 = compute, 1 = do not)	NAREA
k - target element 1 center-point x-coordinate	TGT(1,1)

	Variable
l - target element 1 center-point y-coordinate	TGT(1,2)
m - target element 1 azimuth of primary axis (orientation)	TGT(1,3)
n - target element 1 length	TGT(1,4)
o - target element 1 width	TGT(1,5)
p - surface code (0 = non-surface, 1 = surface)	ITGT(1,1)
q - target element 1 hardness code (crater table index)	ITGT(1,2)
r - target group of target element 1	ITGT(1,3)
s - minimum clear length (MCL) for target element 1	CRIT(1,1)
t - minimum clear width (MCW) for target element 1	CRIT(1,2)
u - number of TOL surfaces	NCP
v - number of taxi surfaces	LV
w - number of attack patterns defined	NPATT
x - number of weapons in pattern	IPAT(1,1)
y - number of submunitions in weapon	IPAT(1,2)
z - weapon code (crater table index number identifying weapon)	IPAT(1,3)
A - pattern shape (0=gen purpose,1=rec CBU,2=elip CBU,3=guided)	IPAT(1,4)
B - range delivery error standard deviation	PATT(1,1)
C - deflection delivery error standard deviation	PATT(1,2)
D - range ballistic error standard deviation	PATT(1,3)
E - deflection ballistic error standard deviation	PATT(1,4)
F - CBU half-pattern length	PATT(1,5)
G - CBU half-pattern width	PATT(1,6)
H - CBU half-void length	PATT(1,7)
I - CBU half-void width	PATT(1,8)
J - munition fuze reliability	PATT(1,9)
K - submunition fuze reliability	PATT(1,10)
L - munition range coordinate relative to center of stick	PATT(1,11)
M - munition deflection coordinate relative to center stick	PATT(1,12)
N - total number of hardness codes	NHARD
O - number of weapons defined	NWPN
P - hardness 1, weapon 1 deny-TOL crater size	CRTAB(1,1,1)
Q - hardness 1, weapon 2 deny-TOL crater size	CRTAB(1,2,1)
R - maintenance patches available	MXPTCH
S - repair priority code	IREPR
T - number of attack passes to be flown	NPASS
U - aimpoint x-coordinate for pass 1	PASS(1,1)
V - aimpoint y-coordinate for pass 1	PASS(1,2)
W - attack heading for pass 1	PASS(1,3)
X - probability of arriving at release point (1st time)	PASS(1,4)
Y - probability of arriving at release point (2nd time)	PASS(1,5)
Z - attack pattern to be flown (PASS matrix index)	IPASS(1,1)
aa - next pass this aircraft will fly	IPASS(1,2)
bb - target element number being attacked	NPX(1)

APPENDIX E

AAPMOD SAMPLE OUTPUT

This appendix describes AAPMOD output. A typical listing is shown on the next page, followed by a brief description. This particular output was produced by the TURBO PASCAL version of AAPMOD.

nsamp = 200[A]   conf interval for 99% level = 1.245 for tgt elt = 1[B]
conf interval for 90% level = 0.795

tgt element	1	2
exp no. hits	41.420[C]	7.805[D]
sigma	6.833[E]	2.093[F]
tgt gp. no.	1	1

target groups

tgt gp. no.	1
exp no. hits	49.225[G]
sigma	7.272[H]

for runways and major taxiways

tgt elt	mcl	mcw	prob cut	sigma	exp no craters	sigma	exp area fill	sigma	exp no filled	sigma	exp appr no crat	sigma	exp appr fill	sigma
1	4000[I]	50[J]	0.750[K]	0.031[L]	2.580[M]	2.265[N]	N/A[O]	N/A[P]	N/A[Q]	N/A[R]	N/A[S]	N/A[T]	N/A[U]	N/A[V]
2	4000	50	0.950	0.015	2.645	1.240	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

combined probabilities of cut

tgt elts	mcl	mcw	prob cut	sigma	exp no craters	sigma	exp area fill	sigma	distribution minimum craters elt 1	elt 2	elt 3	exp appr no crat	sigma	exp appr fill	sigma
1&2[W]	4000	50	0.710	0.032	1.605	1.352	N/A	N/A	0.370[X]	0.340	N/A	N/A	N/A	N/A	N/A

A - number of samples used to calculate report
B - confidence half-interval for indicated target element. AAPMOD picks the target element with the largest number of hits for this portion of the report.
C - expected number of hits for target element number 1
D - expected number of hits for target element number 2
E - standard deviation for expected number of hits on target 1
F - standard deviation for expected number of hits on target 2
G - expected number of hits for target group 1
H - standard deviation for expected number of hits on target group 1
I - minimum clear length for target element number 1
J - minimum clear width for target element number 1
K - probability of cut (probability of denying a minimum clear strip) for target element number 1
L - standard deviation for item K
M - expected number of craters to be repaired in order to re-open a minimum clear strip on target element 1
N - standard deviation for item M
O - expected area to fill in order to re-open a minimum clear strip on target element 1
P - standard deviation for item O
Q - not used by AAPMOD
R - not used by AAPMOD
S - expected number of craters to be repaired in order to clear an approach to a minimum clear strip.
T - standard deviation for item S
U - expected area to fill in order to clear an approach to a minimum clear strip.
V - standard deviation for item U
W - summary statistics for combinations of runways
X - the distribution of minimum number of craters sum to give a probability of cut for a particular runway combination. In this example, 0.370 and 0.340 sum to give the 0.710 figure shown for the combined probability of cut. The distribution of minimum craters gives the relative percentage of time that a particular target element must be repaired to open a minimum clear strip. Recall that if the field is closed, only one minimum clear strip must be re-opened.

APPENDIX F

PASCAL RANDOM NUMBER GENERATOR TEST RESULTS

This appendix provides the source code listings for three of the PASCAL programs used to test Borland International's TURBO PASCAL random number generator and the results of the tests. All of the tests were accomplished with with a 16-bit microcomputer, an 8087 numeric coprocessor unit, and the TURBO-87 Version 2.10A (8087-compatible) TURBO PASCAL compiler.

The following program was used to test for the period of the TURBO PASCAL random number generator. The program begins by initiating the random number generator (randomize), drawing a random number, and assigning this value to a variable (rn). The program then continues to draw random numbers, update counters, and display how many millions of random numbers have been checked on the screen. The program ends if a new random number comes within 0.0000000001 of the original number stored in variable rn.

The test program was run until 50 million random numbers had been generated, at which time the program was stopped by the user. Therefore, there was no reoccurence of the first number after the next 50 million random number draws within a tolerance of 0.0000000001.

```
{==========================================================================}
program RandomNumberPeriod ; { 4 Jan 85 }
{==========================================================================}
var  r, rn : real ;
     count, millions, thousands : integer ;
begin
clrscr ;
writeln( 'Random Number Period Test Program' ) ; writeln ; writeln ;
count := 0 ; millions := 0 ; thousands := 0 ; { initialize counters }
randomize ; { initialize random number generator }
rn := random ; { random produces a random number between 0 and 1 }
repeat
  count := count + 1 ;
  if count = 1000 then begin
    count := 0 ;
    thousands := thousands + 1 ;
    if thousands = 1000 then begin
      thousands := 0 ;
      millions := millions + 1 ;
      gotoxy( 0, 3 ) ;
      clreol ;
      write( millions, ',000,000 numbers checked.' ) ;
    end ;
  end ;
  r := random ; { draw new number and store in variable r }
until ( abs( r - rn ) < 1e-10 ) or ( millions = maxint ) ;
end.
```

The following program conducts a chi-square test to determine whether the random numbers drawn from the TURBO PASCAL random number generator are uniformly distributed. The model for the test is shown in Banks and Carson (3:271). The test is programmed for the observations to be stored in one of 100 bins. The test ends when any bin becomes full: 500 observations maximum per bin. The test was run three times, with the results as shown after the program listing. At a level of significance of alpha = 0.10, the null hypothesis that the numbers were not uniformly distributed would be rejected if the chi-square test statistic was greater than 118.5. All of the tests produced a chi-square value less than 118.5, thus indicating that the null hypothesis could not be rejected. The high level of significance, 0.10, gives more power to the rejection of the null hypothesis, and hence more credence to the conclusion that the random number streams analysed were uniformly distributed.

```
{==========================================================================}
program RandomNumberTests ; { 4 Jan 85 }
{==========================================================================}
var  count, mu, r, rn, ss, sumx, sumxsqr, sx, sxn, x : real ;
     i, icount, j, thousands : integer ;
     period, quit : boolean ;
     freq : array[1..100] of integer ;
     lst : text ;

procedure diskoutput ;
begin
  assign( lst, 'M:BINS100.OUT' ) ;
  rewrite( lst ) ;
end ;

begin { main program }
clrscr ;
diskoutput ;
writeln( 'Random Number Test Program' ) ; writeln ; writeln ;
writeln( lst, 'TURBO PASCAL Random Number Test Program' ) ;
writeln( lst ) ; writeln( lst ) ;
quit := false ; period := false ;
```

```
for i := 1 to 100 do freq[i] := 0 ;  { zero array values }
count := 0 ;  icount := 0 ;  thousands := 0 ;  { initialize counters }
randomize ;  { initialize random number generator }
rn := random ;  { random gives a random number between 0 and 1 }
r := rn ;
repeat
  count := count + 1 ;
  icount := icount + 1 ;
  i := trunc( r * 100 ) + 1 ;  { convert to integer between 1 and 100 }
  freq[i] := freq[i] + 1 ;
  if freq[i] = 500 then quit := true ;
  if icount = 1000 then begin
    icount := 0 ;
    thousands := thousands + 1 ;
    gotoxy( 0, 3 ) ;
    clreol ;
    write( thousands, ',000 numbers checked.' ) ;
  end ;
  r := random ;
  if abs( r - rn ) < 1e-7 then period := true ;  { check for period }
until quit or period ;
write( lst, 'The period of this generator ' ) ;
if period then write( lst, 'equals' ) else write( lst, 'exceeds' ) ;
writeln( lst, ' the amount of numbers generated.' ) ;
writeln( lst, '( Epsilon = 0.0000001 )' ) ;
writeln( lst ) ;
writeln( lst ) ;
writeln( lst, 'Number of random numbers generated  = ', count:5:0  ) ;
writeln( lst ) ;
writeln( lst ) ;
mu := count / 100 ;
sumxsqr := 0 ;
for i := 1 to 100 do sumxsqr := sumxsqr + sqr( freq[i] - mu ) ;
writeln( lst, 'Mu = ', mu ) ;
writeln( lst ) ;
writeln( lst, 'Sum of ( Xi Squared ) = ', sumxsqr ) ;
writeln( lst ) ;
writeln( lst, 'Chi squared = ', sumxsqr / mu ) ;
writeln( lst ) ;
writeln( lst ) ;
writeln( lst ) ;
writeln( lst, 'Contents of bins from 1 to 100.  ( Row 1 = Bins 1-5 )' ) ;
writeln( lst ) ;
for i := 0 to 19 do begin
  for j := 1 to 5 do write( lst, freq[ i * 5 + j ]:10 ) ;
  writeln( lst ) ;
end ;
close( lst ) ;
end.
```

The results of test #1, 100 bins:

TURBO PASCAL Random Number Test Program

The period of this generator exceeds the amount of numbers generated.
(Epsilon = 0.0000001)

Number of random numbers generated = 45481

Mu = 454.81

Sum of (Xi Squared) = 47,765.39

Chi squared = 105.02

Contents of bins from 1 to 100:

Bins 1 to 5:	446	476	445	480	486
Bins 6 to 10:	434	436	476	434	493
Bins 11 to 15:	478	497	426	445	463
Bins 16 to 20:	446	450	460	434	447
Bins 21 to 25:	481	438	437	433	472
Bins 26 to 30:	461	431	441	469	471
Bins 31 to 35:	456	450	475	482	457
Bins 36 to 40:	446	488	464	427	415
Bins 41 to 45:	446	487	456	472	407
Bins 46 to 50:	469	450	461	437	460
Bins 51 to 55:	439	453	486	421	498
Bins 56 to 60:	424	450	481	450	451
Bins 61 to 65:	449	463	477	464	441
Bins 66 to 70:	500	417	495	443	479
Bins 71 to 75:	430	416	419	445	487
Bins 76 to 80:	452	466	462	491	451
Bins 81 to 85:	448	460	434	438	426
Bins 86 to 90:	455	453	438	432	437
Bins 91 to 95:	442	428	454	489	494
Bins 96 to 100:	465	426	462	448	461

The results of test #2, 100 bins:

TURBO PASCAL Random Number Test Program

The period of this generator exceeds the amount of numbers generated.
(Epsilon = 0.0000001)

Number of random numbers generated = 43826

Mu = 438.26

Sum of (Xi Squared) = 50,711.24

Chi squared = 115.71

Contents of bins from 1 to 100:

Bins 1 to 5:	447	429	494	470	413
Bins 6 to 10:	486	446	410	406	462
Bins 11 to 15:	467	461	417	429	420
Bins 16 to 20:	432	444	453	434	406
Bins 21 to 25:	432	477	434	476	438
Bins 26 to 30:	376	439	415	440	474
Bins 31 to 35:	433	425	500	455	453
Bins 36 to 40:	457	429	415	431	415
Bins 41 to 45:	414	462	434	419	440
Bins 46 to 50:	437	463	416	416	438
Bins 51 to 55:	471	439	454	461	451
Bins 56 to 60:	421	424	449	418	439
Bins 61 to 65:	437	445	456	456	462
Bins 66 to 70:	429	432	409	443	471
Bins 71 to 75:	431	418	410	449	436
Bins 76 to 80:	404	454	434	486	435
Bins 81 to 85:	454	404	431	421	410
Bins 86 to 90:	424	448	440	436	415
Bins 91 to 95:	400	441	426	458	473
Bins 96 to 100:	410	435	428	418	451

The results of test #3, 100 bins:

TURBO PASCAL Random Number Test Program

The period of this generator exceeds the amount of numbers generated.
(Epsilon = 0.0000001)

Number of random numbers generated = 45021

Mu = 450.21

Sum of (Xi Squared) = 45,602.59

Chi squared = 101.29

Contents of bins from 1 to 100:

Bins 1 to 5:	414	486	457	445	500
Bins 6 to 10:	437	417	442	403	400
Bins 11 to 15:	467	430	420	427	459
Bins 16 to 20:	440	432	443	455	449
Bins 21 to 25:	470	450	467	452	454
Bins 26 to 30:	442	437	448	451	448
Bins 31 to 35:	451	429	440	495	423
Bins 36 to 40:	444	490	453	461	460
Bins 41 to 45:	457	449	449	402	460
Bins 46 to 50:	485	453	461	443	448
Bins 51 to 55:	433	430	434	472	400
Bins 56 to 60:	440	431	475	470	434
Bins 61 to 65:	438	471	429	429	477
Bins 66 to 70:	441	433	472	495	442
Bins 71 to 75:	400	453	426	451	432
Bins 76 to 80:	460	474	408	449	496
Bins 81 to 85:	460	469	443	429	437
Bins 86 to 90:	407	432	457	464	417
Bins 91 to 95:	438	435	451	471	424
Bins 96 to 100:	472	451	474	424	448

Another chi-square test was accomplished using 1024 bins, a maximum of 32,767 observations per bin, and a smaller period tolerance level of 0.0000000001. The chi-square value obtained was divided by the number of bins, 1024, to obtain a test statistic. The resultant value would be indicative of uniformly distributed numbers if it were in the range 0.5 to 2.0 (56:446). For each of several tests performed, the test statistics fell within the range indicative of uniformly distributed numbers. The following shows the program used and results for one of the test runs.

```
{=====================================================================}
program RandomNumberTests ; { 4 Jan 85 }
{=====================================================================}
var  count, mG, r, rn, ss, sumx, sumxsqr, sx, sxn, x : real ;
     i, icount, j, millions, thousands : integer ;
     period, quit : boolean ;
     freq : array[1..1024] of integer ;

     lst : text ;

procedure printrow ;
begin
  for j := 1 to 8 do write( lst, freq[ i X 8 + j ]:9 ) ;
  writeln( lst ) ;
end ;

procedure diskoutput ;
begin
  assign( lst, 'M:RNTESTS.OUT' ) ;
  rewrite( lst ) ;
end ;

begin { main program }
clrscr ;
diskoutput ;
writeln( 'Random Number Test Program' ) ; writeln ; writeln ;
writeln( lst, 'TURBO PASCAL Random Number Test Program' ) ;
writeln( lst ) ; writeln( lst ) ;
quit := false ; period := false ;
for i := 1 to 1024 do freq[i] := 0 ;
count := 0 ; icount := 0 ; thousands := 0 ; millions := 0 ;
randomize ;
rn := random ; r := rn ;
```

```
repeat
  count := count + 1 ;
  icount := icount + 1 ;
  i := trunc( r * 1024 ) + 1 ;
  freq[i] := freq[i] + 1 ;
  if freq[i] = maxint then quit := true ;
  if icount = 1000 then begin
    icount := 0 ;
    thousands := thousands + 1 ;
    if thousands = 1000 then begin
      thousands := 0 ;
      millions := millions + 1 ;
      gotoxy( 0, 3 ) ;
      clreol ;
      write( millions, ',000,000 numbers checked.' ) ;
    end ;
  end ;
  r := random ;
  if abs( r - rn ) < 1e-10 then period := true ;
until quit or period ;
write( lst, 'The period of this generator ' ) ;
if period then write( lst, 'equals' ) else write( lst, 'exceeds' ) ;
writeln( lst, ' the amount of numbers generated.' ) ;
writeln( lst ) ;
writeln( lst ) ;
writeln( lst, count, ' random numbers were generated.' ) ;
writeln( lst ) ;
writeln( lst ) ;
mu := count / 1024 ;
sumx := 0 ;
sumxsqr := 0 ;
for i := 1 to 1024 do sumxsqr := sumxsqr + sqr( freq[i] - mu ) ;
writeln( lst, 'Mu = ', mu ) ;
writeln( lst ) ;
writeln( lst, 'Sum of ( Xi Squared ) = ', sumxsqr ) ;
writeln( lst ) ;
writeln( lst, 'Chi squared = ', sumxsqr / mu ) ;
writeln( lst, ^L'Contents of bins from 1 to 1024.  ( Row 1 = Bins 1-8 )' ) ;
writeln( lst ) ;
for i := 0 to 49 do printrow ;
writeln( lst, ^L ) ;
for i := 50 to 99 do printrow ;
writeln( lst, ^L ) ;
for i := 100 to 127 do printrow ;
close( lst ) ;
end.
```

The results of test #1, 1024 bins:

TURBO PASCAL Random Number Test Program

The period of this generator exceeds the amount of numbers generated.
(Epsilon = 0.0000000001)

Number of random numbers generated = 32,986,943

Mu = 32,213.81

Sum of (Xi Squared) = 31,529,942.62

Chi squared = 978.770941187245E+002

Test statistic = 0.956

Contents of bins from 1 to 1024. (Row 1 = Bins 1-8)

Row								
1	31912	32455	31819	32102	31952	32054	32246	32081
2	31864	32105	32307	32323	32245	32394	32144	32139
3	32767	32231	32238	32062	32200	32269	32264	32173
4	32251	31865	32344	32098	31944	32263	32409	32175
5	32305	32164	31841	32382	32280	32191	32026	32317
6	32121	32395	32475	32282	32283	32170	32113	32282
7	32355	32365	32230	32252	32071	32224	32165	31956
8	32438	32492	31920	32236	32306	32358	32182	31981
9	32085	31902	32028	32122	32172	31769	32492	32110
10	32190	32193	32393	31988	32271	32073	32128	32041
11	32363	32283	32207	32409	32046	32111	32279	32250
12	32303	32019	32152	32404	32028	32418	32117	32303
13	32660	32197	32090	32441	32224	32081	32142	32377
14	32123	32316	32088	32143	31974	32264	32009	32044
15	32485	32194	32337	32263	32236	32628	32090	32077
16	32278	32082	32325	32035	31926	32165	32327	32274
17	32133	32448	32147	32487	32266	32439	32272	32236
18	32398	32448	32089	32323	32162	32075	32288	32378
19	32073	32024	32339	32421	32253	32241	32238	32200
20	32102	32127	32191	32072	32133	32243	31911	31949
21	32194	31977	32238	32003	32350	31973	32307	32056
22	32061	32022	32226	32194	32061	32421	32309	32191
23	31951	31930	32361	32086	32372	32151	32154	32043
24	32348	32386	31943	32372	32477	32061	32329	32260
25	32258	32000	32287	31895	32307	31992	32088	32049
26	32447	32322	32263	32529	32145	32271	32082	32121
27	32243	32145	32466	32384	32375	32183	32436	32530
28	32355	32509	32555	32058	32217	32532	32231	32476

Row								
29	32357	32060	32190	32319	31980	32083	32118	32286
30	31883	32452	32518	32385	32297	32210	31962	32176
31	31965	32192	32140	32093	32408	32284	32296	32381
32	32564	32358	32563	32536	32018	32433	32214	32396
33	32175	32360	32049	32308	32077	32192	32594	32390
34	32169	32110	32253	32358	32198	32222	32559	32145
35	32344	32193	32267	31924	32343	31776	32475	32242
36	32107	32192	32175	32092	32188	32232	32438	31954
37	32386	31962	32114	32116	32112	32031	32153	32375
38	32536	31916	32359	32097	32094	32296	32231	32198
39	31874	32318	32562	32289	31978	32095	32048	32230
40	32184	32064	32046	32721	32429	32210	32284	32032
41	32167	32495	32120	31829	32100	31983	32325	32112
42	32127	32411	31914	32336	32117	31738	31924	32100
43	32334	31939	32281	32522	31932	32216	31931	32423
44	32538	32168	32203	31974	32382	32298	32185	32127
45	31901	31757	32232	32112	32049	32259	32432	32299
46	32220	32216	32092	32240	32267	31990	32238	32244
47	32226	32286	32142	31904	32127	32115	32242	32294
48	31967	32380	32333	32417	32605	32102	31921	32206
49	32308	32476	32192	32233	32507	32303	32133	32132
50	32149	32093	32138	32133	32319	32375	32166	32151
51	32063	31884	32078	32195	32169	32214	31851	32043
52	32578	32495	32157	32267	32197	32191	32369	32436
53	32589	32304	32051	32429	32291	31994	32078	32216
54	32391	32234	32566	32250	32143	32302	32322	32554
55	31996	32113	31920	32083	32406	31862	32222	32232
56	32421	32115	32290	31928	32199	32125	32061	32373
57	32307	32011	32288	32112	31747	32179	32314	32189
58	32224	32240	32393	32393	32603	31937	32204	31825
59	32215	32440	32271	31945	32315	32271	32304	32299
60	32459	32252	32228	32488	32007	32072	32165	32229
61	32184	32261	32117	32423	32036	32510	32091	32425
62	32384	32120	32371	32420	32333	31863	32157	32155
63	32189	31935	32235	32102	32269	32471	31960	31958
64	32110	32217	32130	32100	32126	32278	32042	32682
65	32394	32708	32210	32535	32253	31776	32204	32091
66	32255	32398	32149	31969	32344	32063	32080	31911
67	32113	32122	32346	32684	32012	32227	32080	32401
68	32002	32184	32152	32540	31658	32417	32278	32042
69	32261	32234	32393	32126	32322	32415	32194	32354
70	32251	32365	32162	32223	32393	32175	32460	32280
71	32431	32274	32429	32452	32137	32211	32725	32219
72	31849	31997	31979	32165	32563	32129	32478	32525
73	32315	32408	32381	32078	32202	32072	32052	32209
74	32151	31751	32328	31968	32318	32337	32282	32171
75	32348	32228	32185	32459	32251	32062	32518	32249
76	31923	32014	32211	31995	32406	32066	31819	32023
77	31950	32260	32174	32149	32235	32083	32433	32025
78	32172	32237	32435	32325	32266	32330	32189	32225
79	32466	32055	32071	32058	32231	32302	32245	32103
80	31960	32234	32447	32047	32391	32233	32269	32355

Row								
81	32166	32163	32165	32042	32300	32119	32214	32205
82	32483	32056	32116	32227	32251	32443	32350	32289
83	32446	31856	32075	32453	32083	31874	31965	32171
84	32115	32238	32085	32162	31880	32374	32153	32077
85	32126	32238	32465	32189	32193	32460	32348	32348
86	31901	32319	32400	31994	32310	32161	32342	32154
87	32246	32283	32180	31933	32295	32103	32257	32188
88	32376	32389	32165	31861	31976	32037	32331	32204
89	31875	32281	32413	32043	32082	32306	32207	32264
90	32553	32337	32175	32311	32147	32084	32318	32093
91	32262	32280	32174	32294	32160	32252	32322	32221
92	32264	32127	32112	32201	32353	32296	32049	31981
93	32481	32459	32446	32250	32436	32184	32149	32045
94	32091	32131	32254	32153	32008	31832	32119	31971
95	32231	32490	32183	32046	32050	32215	32250	32285
96	31954	32058	32492	32272	32279	31882	32170	32345
97	32213	32192	32100	31943	32014	32222	31967	32005
98	32397	32137	32495	32265	31873	32239	31956	32025
99	32072	32424	32028	32147	32193	32403	32078	32275
100	32333	32331	32279	32230	32163	32124	32229	32249
101	31976	32512	32313	32051	32032	32327	32182	32202
102	32467	32398	32535	32189	32345	32258	32196	32226
103	32560	32339	32303	32272	32566	31939	31945	32040
104	32025	32158	32151	32228	32313	32462	32504	32076
105	32090	32142	31847	32160	32065	32314	32250	32386
106	32434	32313	32346	32185	32000	32134	32374	32344
107	32209	32064	32061	32043	32104	31863	31935	32266
108	32147	31812	32482	32105	32267	32590	32088	32238
109	32313	32123	32148	32135	32101	32091	32164	32361
110	32224	31972	32177	31967	32261	32175	32213	32329
111	32135	32164	32270	32025	32384	32161	32314	32115
112	32096	32130	32207	32272	32256	32302	32344	32337
113	32303	32063	32064	32384	32573	32348	32605	32379
114	32243	32118	32113	32118	32218	32032	32396	32531
115	32316	32218	32508	32324	32277	32445	32159	31896
116	32191	32321	32359	32325	32249	32250	32202	32122
117	31955	32247	32532	32295	32461	32255	32045	32260
118	32200	32359	32320	32096	32330	32356	32500	32163
119	32255	32068	32650	32282	32352	32144	32280	32074
120	32216	32198	32411	32328	32497	32183	32601	32235
121	31916	32221	32427	32298	32354	32282	32474	32122
122	32236	32082	32039	32064	32278	31963	32429	32023
123	32099	32110	32005	32518	32288	32504	32343	32093
124	32132	32210	32165	32209	32334	32197	32225	32324
125	32313	32283	32210	31940	32399	32299	32221	32199
126	31764	32376	32518	32671	32182	32230	32044	32406
127	32249	32191	32201	32179	32358	32054	32340	32384
128	31993	32450	31891	32171	32261	32271	32532	32000

APPENDIX G

PASCAL LISTING FOR RESPONSE SURFACE METHODOLOGY

This appendix contains TURBO PASCAL computer code for performing regression and analyzing response surfaces. The main file, RESPONS1.PAS, is self-documenting with a short description of the program as well as required inputs and various options. Two separate files are contained in this appendix. As described at Appendix A, large TURBO PASCAL programs are most easily compiled by using "Include" files. RESPONSE.PAS contains only one line of code which "includes" the main response surface methodology program located in RESPONS1.PAS. RESPONS1.PAS is too large to be compiled directly on 64K machines in TURBO PASCAL. The programs listed in this appendix are:

Section 1: RESPONSE.PAS

Section 2: RESPONS1.PAS

```
{==========================================================}
{============== FILE RESPONSE.PAS    17 Jan 85 ============}
{==========================================================}

{$IRESPONS1.PAS  compiler directive to include file RESPONSE.PAS}
```

```
{==========================================================}
{============== FILE RESPONS1.PAS    17 Jan 85 ============}
{==========================================================}

program Response ;    { 17 January 1985, Maj David Roodhouse }

(* Program Description:  This program performs linear regression on
 *  input data.  One dependent variable (y) and up to 16 independant
 *  variables (x's) are permitted in the present configuration.  This
 *  includes any second-order terms, so the largest second-order problem
 *  can include up to 4 first-order variables, 4 squared terms, 6
 *  interaction terms, and the mean for a total of 15.  The largest
 *  first-order problem can include 4 main effects, six 2-factor
 *  interaction terms, four 3-factor interaction terms, one 4-factor
 *  interaction term, and the mean for a total of 16.  This can be
 *  easily expanded in the declaration and initialization sections.
 *  Note that memory becomes an increasing problem with larger designs.
 *  Up to 50 data points can be entered presently.
 *
 *    Input data can either be placed in an input disk file (for which
 *  you will be prompted) or entered directly.  The disk file is
 *  encouraged since there is no provision for correcting errors when
 *  entering data directly.
 *
 *    A typical input disk file would appear as follows:
 *
 *         8
 *         7
 *         0.650   -1  -1  -1   1   1   1  -1
 *         0.140    1  -1  -1  -1  -1   1   1
 *         0.795   -1   1  -1  -1   1  -1   1
 *         0.030   -1  -1   1   1  -1  -1   1
 *         0.875    1   1  -1   1  -1  -1  -1
 *         0.345    1  -1   1  -1   1  -1  -1
 *         0.020   -1   1   1  -1  -1   1  -1
 *         0.450    1   1   1   1   1   1   1
 *
 *  where 8 is the number of observations , 7 is the number of effects
 *  NOT including the mean, the left-most column contains the dependent
 *  variables values for each observation, and the remainder represents
 *  the design matrix.  In this case, the design is for 3 factors with
 *  full factorial (columns 2-4), three 2-factor interaction terms
 *  (columns 5-7), and the single 3-factor interaction term (column 8).
 *
 *    The program expects data in the form of an equation:
 *      y = b0 + b1*x1 + b2*x2 + b3*x1^2 + b4*x2^2 + b5*x1*x2
 *
 *  The y (dependent variable) and x's (independent variables) are entered
 *  in order.  The program uses least squares to figure the b's.
 *
```

```
%    The program uses matrix manipulations (Gram-Schmidt Orthogonal-
%  ization to solve for the regression coefficients (b's).  This is
%  done as follows:
%
%        1.  Read input data (X and Y matrices), from disk or Keyboard
%        2.  Break the X matrix into 2 component matrices ( X = QR)
%             a.  Q is orthogonal (greatly simplifies matrix ops)
%             b.  R is upper triangular (easy to back solve)
%        3.  Back solve for beta's (b's or regression coefficients)
%             a.  This amounts to solving ( RXb = Q'y)
%        4.  Compute variance and covariance matrices
%        5.  Write the results to the screen
%
%    The program employs more classic Gauss-Jordan matrix inversion
%  for finding response surface stationary points if this option
%  is selected by answering "y" when prompted.
%
%    ANOVA is performed with the proper response to the prompt.
%  More than one center point is required to perform ANOVA in this program
%  and the center points must be the last lines in the input file.
%  The multiple center points are used to estimate variance in the data.
%  More than one center point enables the ANOVA.  At this point the
%  ANOVA tests how well the model fits the data and how significant
%  lack of fit is.  When prompted, an entry of "1" supresses the ANOVA
%  since only one center point is implied.  Use this entry even if there
%  no center points included in the design.  Any other positive number
%  enables the ANOVA and must correspond to the number of center points.
%
%    Finally, stationary point calculation are made if selected.
%  You should only use this section when employing a second order model
%  or the results will be unpredictable.  The x's for the stationary
%  point are the coordinate of a local minimum, maximum, or saddle point.
%  The y value is the dependent variable value at the stationary point.
%)

  type  ( T Y P E   D E C L A R A T I O N S )

matrix1 = array[1..4,1..4] of real ;

  var  ( V A R I A B L E   D E C L A R A T I O N S   A N D   K E Y )

( the program is presently configured for up to 50 observations
    and up to 16 explanatory variables; be sure to update the array
    declarations AND procedure Initialize when expanding arrays as
    well as procedure EnterData for error trapping.  All the sections
    that need to be changed can be found by searching for "update point" )

diskfile : text ;              (disk file variable)

fname : string[14] ;           (for file names)

Beta, BetaInverse : matrix1 ;  (cross product correlation coefficients)
```

```
b,                            {regression coefficients}
c,                            {intermediate matrix}
XprimeYbar,                   {matrix intermediate product}
Xcrit                         {response surface stationary point}
  : array[1..16] of real ;    {update point}

e,                            {error matrix}
y                             {dependent variable values}
  : array[1..58] of real ;    {update point}

r,                            {component matrix X = QR}
v,                            {variance matrix}
w                             {unscaled variance-covariance matrix}
  : array[1..16,1..16] of real ; {update point}

q,                            {orthogonal component matrix X = QR}
x                             {independent variable values}
  : array[1..58,1..16] of real ; {update point}

BhatXprimeYbar,               {uncorrected regression sum of squares}
ev,                           {error variance}
MSE,                          {mean square - error}
MSFit,                        {mean square - lack of fit}
MSR,                          {mean square - regression}
SSE,                          {sum of squares - error}
SSFit,                        {sum of squares - lack of fit}
SSR,                          {sum of squares - regression}
SST,                          {sum of squares - total}
SSTcorrected,                 {sum of squares - total, corrected for mean}
sumE,                         {sum of absolute errors - center point}
s,                            {intermediate variable}
ycrit                       {dependent variable value at stationary point}
  : real ;

i,j,l,m,z,                    {loop counters}
f,                            {number of main effects}
k,                            {number of independent variables}
n,                            {number of observations}
p                             {number of center points}
  : integer ;

solveXo,          {flag for solving response surface stationary point}
anova,                        {flag to perform anova}
print,                        {hard-copy flag}
continue            {cross product matrix nonsingular - can continue}
   : boolean ;

choice,                       {hard-copy option}
choice2                       {solve stationary point option}
   : char ;
```

```
{==================================================================}
procedure pause ;
(*
*   delays crt output until a key is pressed
*)
begin
  writeln ;
  write( '        p r e s s   a n y   k e y   t o   c o n t i n u e  ' ) ;
  repeat until keypressed ;
  writeln ;
end ;  (* procedure pause *)

{==========================================================}
procedure Processfile ;

{      Initial processing of filename entered by user }

  var p : integer ;
begin
  writeln ;
  write( 'Enter name for the regression input file ==> ' ) ;
  readln( fname ) ;
  writeln ;
  p := pos( '.', fname ) ;           {filetype can be 3 characters max}
  if p <> 0 then delete( fname, p + 4, 14 ) ;
  for p :=1 to 14 do fname[p] := upcase( fname[p] ) ;
  repeat                             {delete leading blanks}
    p := pos( ' ', fname ) ;
    if p = 1 then delete( fname, p, 1 ) ;
  until p <> 1 ;
  writeln ;
end ; {procedure Processfile}
```

```
{=================================================================}
procedure Initialize ;

(*  Initializes various arrays and variables used throughout program  *)

begin
  writeln ;
  writeln( 'Initializing . . .' ) ;
  writeln ;
  n := 50 ;       {update point} {up to 50 observations}
  k := 16 ;       {update point} {16 explanatory variables including mean}
  anova := false ;
  solveXo := false ;
  continue := false ;
  for i := 1 to k do begin
    b[i] := 0 ;
    c[i] := 0 ;
    for j := 1 to k do begin
      r[i,j] := 0 ;
      v[i,j] := 0 ;
      w[i,j] := 0 ;
    end ;
  end ;
  for i := 1 to n do begin
    e[i] := 0 ;
    y[i] := 0 ;
    for j := 1 to k do begin
      q[i,j] := 0 ;
      x[i,j] := 0 ;
    end ;
  end ;
end ;  {procedure Initialize}
```

```
{=====================================================================}
procedure EnterData;

begin
  writeln ;
  writeln( 'Least squares estimation using Gram-Schmidt Orthogonalization' ) ;
  writeln ;
  write( 'How many observations do you have? ' ) ;
  repeat {$I-} readln( n ) {$I+} until ioresult = 0 ;
  writeln;
  if n > 50 then                                  {update point}
    begin
      writeln( 'You must resize arrays for this many observations.' ,
               ' Terminating...' ) ;
      halt ;
    end ;
  write( 'How many explanatory variables (X''s) do you have? ' ) ;
  repeat {$I-} readln( k ) {$I+} until ioresult = 0 ;
  writeln ;
  if k > 16 then                                  {update point}
    begin
      writeln( 'You must resize arrays for this many variables.' ,
               ' Terminating...' ) ;
      halt ;
    end ;
  writeln( 'Please enter equations in order:' ) ;
  writeln( 'For X''s - the first subscript is the equation number,' ) ;
  writeln( '  the second subscript is the variable number.' ) ;
  writeln ;
  for j := 1 to n do
    begin
      write( 'Y(' , j , ')= ? ' ) ;
      repeat {$I-} readln( y[j] ) {$I+} until ioresult = 0 ;
      writeln ;
      for i := 1 to k do
        begin
          write( 'X(' , j , ',' , i , ')= ? ' ) ;
          repeat {$I-} readln( x[j,i+1] ) {$I+} until ioresult = 0 ;
          writeln ;
        end ;
      writeln ;
      x[j,1] := 1.0 ;      {constant term}
    end ;
  writeln ;
end ;  {procedure EnterData}
```

```
{=================================================================}
procedure GetData ;

{   Retrieves data file from disk }

  var badname : boolean ;

begin
  repeat
    processfile ;
    if length( fname ) = 0
      then badname := true
      else
        begin
          assign( diskfile, fname ) ;
          {$I-} reset( diskfile ) ;      {$I+ check if diskfile exists}
          badname := ioresult <> 0 ;
        end ;
    if badname then
      begin
        writeln( 'File < ', fname, ' > does not exist.  Specify the correct ',
                         'drive if' ) ;
        writeln( 'different from the logged drive.  Enter control-C to ',
                 'abort program. ' ) ;
        writeln
      end ;
  until not badname ;
  writeln ;
  writeln( 'R e a d i n g   d i s k   f i l e   < ', fname, ' > ...' ) ;
  writeln ;
  readln( diskfile, n ) ;
  if n > 50 then                  {update point}
    begin
      writeln( 'Too many observations for matrices.  Terminating ...' ) ;
      halt ;
    end ;
  readln( diskfile, k ) ;
  if k > 16 then                  {update point}
    begin
      writeln( 'Too many variables for matrices.  Terminating ...' ) ;
      halt ;
    end ;
  for j := 1 to n do
    begin
      read( diskfile, y[j] ) ;
      for i := 1 to k do
        read( diskfile, x[j,i+1] ) ;
      x[j,1] := 1.0 ;
    end ;
  close( diskfile ) ;
end ;  {procedure GetData}
```

```
{=====================================================================}
procedure FirstCalculations ;

{ Calculate SST and other values before input matrices are modified }

begin
  SST := 0 ;
  for i := 1 to n do
    SST := SST + sqr( y[i] ) ;
  for i := 1 to k+1 do
    begin
      XprimeYbar[i] := 0 ;
      for j := 1 to n do
        XprimeYbar[i] := XprimeYbar[i] + x[j,i] * y[j] ;
    end ;
end ;  {procedure FirstCalculations}

{=====================================================================}
procedure RKeyElement ;

  var rr : real ;

begin
  rr := 0.0 ;
  for i := 1 to n do rr := rr + sqr( x[i,z] ) ;
  r[z,z] := sqrt( rr )
end ;  {procedure RKeyElement}

{=====================================================================}
procedure QColumn ;

begin
  for i := 1 to n do q[i,z] := x[i,z] / r[z,z]
end ;  {procedure QColumn}

{=====================================================================}
procedure RColumn ;

begin
  if z <> k then
    for l := z+1 to k do
      begin
        r[z,l] := 0.0 ;
        for i := 1 to n do r[z,l] := r[z,l] + x[i,l] * q[i,z]
      end ;
end ;  {procedure RColumn}
```

```
{==============================================================}
procedure CElement ;

begin
  c[z] := 0.0 ;
  for i := 1 to n do c[z] := c[z] + y[i] * q[i,z]
end ;  {procedure CElement}

{==============================================================}
procedure ReviseX ;

begin
  if z <> k then
    for i := 1 to n do
      for l := z+1 to k do
        x[i,l] := x[i,l] - q[i,z] * r[z,l]
end ;  {procedure ReviseX}

{==============================================================}
procedure Orthogonalize ;

begin
  writeln( 'Computing . . .' ) ;
  k := k+1 ;
  for z := 1 to k do
    begin
      RKeyElement ;
      QColumn ;
      RColumn ;
      CElement ;
      ReviseX
    end ;
end ;  {procedure Orthogonalize}

{==============================================================}
procedure BackSolve ;  {for regression coefficients}

begin
  b[k] := c[k] / r[k,k] ;
  for i := k-1 downto 1 do
    begin
      s := 0.0 ;                          {left-side sum}
      for j := i+1 to k do s := s + r[i,j] * b[j] ;
      b[i] := ( c[i] - s ) / r[i,i]
    end ;
end ;  {procedure BackSolve}
```

```
{==========================================================================}
procedure ErrorVariance ;        {Vector of residuals (E = Y - QXC)}

begin
  for i := 1 to n do
    begin
      s := 0.0 ;
      for j := 1 to k do s := s + q[i,j] * c[j] ;
      e[i] := y[i] - s
    end ;
  s := 0.0 ;       {Error variance}
  for i := 1 to n do s := s + sqr( e[i] ) ;
  if n <> k then ev := s / (n-k)
    else ev := 0.0 ;
end ;  {procedure ErrorVariance}

{==========================================================================}
procedure InvertMatrixR ;

begin
  for i := 1 to k do v[i,i] := 1 / r[i,i] ;  {Diagonal elements}
  for i := k-1 downto 1 do                    {Off-diagonal elements}
    for j := i+1 to k do
      begin
        s := 0.0 ;                             {Left-side sum}
        for l := i+1 to j do s := s + r[i,l] * v[l,j] ;
        v[i,j] := -s / r[i,i]
      end ;
end ;  {procedure InvertMatrixR}

{==========================================================================}
procedure UnscaledVarCovMatrix ;

begin
  for i := 1 to k do
    for j := 1 to k do
      begin
        w[i,j] := 0.0 ,
        for l := 1 to k do w[i,j] := w[i,j] + v[i,l] * v [j,l]
      end ;
end ;  {procedure UnscaledVarCovMatrix}

{==========================================================================}
procedure VarianceCovariance ;

begin
  ErrorVariance ;
  InvertMatrixR ;
  UnscaledVarCovMatrix
end ;  {procedure VarianceCovariance}
```

```
{==============================================================}
procedure SumOfSquares ;

begin
  writeln ;
  writeln( 'Enter the number of center points run for response surface' ,
          ' analysis.' ) ;
  write( 'Enter 1 if no ANOVA is desired ==> ' ) ;
  repeat {$I-} readln( p ) {$I+} until ioresult = 0 ;
  writeln ;
  if p > 1 then anova := true ;
  if anova then
    begin
      BhatXprimeYbar := 0 ;
      for i := 1 to k do
        BhatXprimeYbar := BhatXprimeYbar + b[i] * XprimeYbar[i] ;
      SSTcorrected := SST - sqr( XprimeYbar[1] ) / n ;
      SSR := BhatXprimeYbar - sqr( XprimeYbar[1] ) / n ;
      SSE := 0 ;
      sumE := 0 ;
      for i := n-p+1 to n do
        begin
          SSE := SSE + sqr( y[i] ) ;
          sumE := sumE + y[i] ;
        end ;
      SSE := SSE - sqr( sumE ) / p ;
      SSFit := SSTcorrected - SSR - SSE ;
      MSR := SSR / (k-1) ;
      MSFit := SSFit / (n-k-p+1) ;
      MSE := SSE / (p-1) ;
    end ;
end ; {procedure SumOfSquares}
```

```
{===========================================================================}
procedure InvertMatrixB( b : matrix1; var c : matrix1 ; r : integer ) ;

  var i,j,k,l : integer ;
      s,t : real ;
      nonsingular : boolean ;
begin
  continue := true ;
  for i := 1 to r do
    for j := 1 to r do
      c[i,j] := 0.0 ;
  for i := 1 to r do
    c[i,i] := 1.0 ;
  j := 0 ;
  repeat
    j := j + 1 ;
    nonsingular := false ;     {reinitializing flag - presumed singular}
    i := j-1 ;
    repeat
      i := i + 1 ;
      if b[i,j] <> 0.0 then nonsingular := true ; {found non-zero value}
    until (i = r) or nonsingular ;
    if not nonsingular then continue := false ;    {singular matrix}
    if continue then
      begin
        for k := 1 to r do
          begin
            s := b[j,k] ;
            b[j,k] := b[i,k] ;
            b[i,k] := s ;
            s := c[j,k] ;
            c[j,k] := c[i,k] ;
            c[i,k] := s ;
          end ;
        t := 1 / b[j,j] ;
        for k := 1 to r do
          begin
            b[j,k] := t * b[j,k] ;
            c[j,k] := t * c[j,k] ;
          end ;
        for l := 1 to r do
          if l <> j then
            begin
              t := -b[l,j] ;
              for k := 1 to r do
                begin
                  b[l,k] := b[l,k] + t * b[j,k] ;
                  c[l,k] := c[l,k] + t * c[j,k] ;
                end ;
            end ;
      end ;
  until (j = r) or not continue ;
end ;  {procedure InvertMatrixB}
```

```
{==============================================================}
procedure StationaryPoint ;

  var  step : integer ;         {Keep track of position in beta array}

begin
  writeln( 'Do you want to make stationary point calculations for' ,
          ' response surface' ) ;
  write( 'analysis (y/n) ==> ' ) ;
  repeat {$I-} readln( choice2 ) {$I+} until ioresult = 0 ;
  writeln ;
  if upcase( choice2 ) = 'Y' then solveXo := true ;
  if solveXo then
    begin
      write( 'Enter the number of main effects ==> ') ;
      repeat {$I-} readln( f ) {$I+} until ioresult = 0 ;
      writeln ;
      step := 1 ;
      for i := 1 to f do
        begin
          Beta[i,i] := b[f+1+i] ;
          if i <> f then
            begin
              for j := i + 1 to f do
                begin
                  Beta[i,j] := b[2*f + 1 + step] / 2 ;
                  Beta[j,i] := Beta[i,j] ;
                  step := step + 1 ;
                end ;
            end ;
        end ;
      InvertMatrixB( Beta, BetaInverse, f ) ;
      clrscr ;
      writeln ;
      writeln( 'Beta matrix follows: ' ) ;
      writeln ;
      for i := 1 to f do
        begin
          for j := 1 to f do
            write( Beta[i,j]:8:4 ) ;
          writeln ;
        end ;
      writeln ;
      writeln ;
      writeln( 'BetaInverse matrix follows: ' ) ;
      writeln ;
      for i := 1 to f do
        begin
          for j := 1 to f do
            write( BetaInverse[i,j]:8:4 ) ;
          writeln ;
        end ;
      pause ;
```

```
      for i := 1 to f do
        begin
          xcrit[i+1] := 0 ;
          for j := 1 to f do
            xcrit[i+1] := xcrit[i+1] - BetaInverse[i,j] * b[j+1] ;
          xcrit[i+1] := xcrit[i+1] / 2 ;
        end ;
      xcrit[1] := 1.0 ;
      for i := f+2 to 2*f + 1 do
        xcrit[i] := sqr( xcrit[i-f] ) ;
      if f = 2 then xcrit[6] := xcrit[2] * xcrit[3] ;
      if f = 3 then
        begin
          xcrit[8] := xcrit[2] * xcrit[3] ;
          xcrit[9] := xcrit[2] * xcrit[4] ;
          xcrit[10] := xcrit[3] * xcrit[4] ;
        end ;
      if f = 4 then
        begin
          xcrit[10] := xcrit[2] * xcrit[3] ;
          xcrit[11] := xcrit[2] * xcrit[4] ;
          xcrit[12] := xcrit[2] * xcrit[5] ;
          xcrit[13] := xcrit[3] * xcrit[4] ;
          xcrit[14] := xcrit[3] * xcrit[5] ;
          xcrit[15] := xcrit[4] * xcrit[5] ;
        end ;
      ycrit := 0 ;
      for i := 1 to k do
        ycrit := ycrit + xcrit[i] * b[i] ;
    end ;
end ;  {procedure StationaryPoint}
```

```
{===========================================================================}
procedure DisplayResults ;

begin
  print := false ;
  writeln ;
  write( 'Do you want hard-copy (y/n) ==> ' ) ;
  repeat {$I-} readln( choice ) {$I+} until ioresult = 0 ;
  writeln ;
  if upcase( choice ) = 'Y' then print := true ;
  clrscr ;
  writeln( 'Estimated constant term and regression coefficients:' ) ;
  if print then writeln( lst, 'Estimated constant term and regression ',
                         'coefficients:' ) ;
  writeln ;
  if print then writeln( lst ) ;
  for i := 1 to K do
    begin
      writeln( 'B(' , i-1 , ')= ' , b[i] ) ;
      if print then writeln( lst , 'B(' , i-1 , ')= ' , b[i] )
    end ;
  writeln ;
  if print then writeln( lst ) ;
  pause ;
  clrscr ;
  writeln ;
  writeln( 'Estimated standard errors:' ) ;
  writeln ;
  if print then begin
    writeln( lst ) ;
    writeln( lst , 'Estimated standard errors:' ) ;
    writeln( lst )
  end ;
  for i := 1 to K do begin
    writeln( 'SE(' , i , ')= ' , sqrt( ev * w[i,i] ) ) ;
    if print then writeln( lst , 'SE(' ,i, ')= ' ,sqrt( ev * w[i,i] ))
  end ;
  pause ;
```

```
if anova then
  begin
    clrscr ;
    writeln ;
    writeln( 'A N A L Y S I S   O F   V A R I A N C E' ) ;
    writeln ;
    writeln( '  Source         SS         d.f      MS        F' ) ;
    writeln( '========      =======     ====  ========   =======' ) ;
    writeln ;
    writeln( 'Regression ', SSR:12:4, (k-1):8, MSR:12:4, (MSR * (n-k)
              / (SSE + SSFit) ):12:4 ) ;
    writeln ;
    writeln( 'Lack of fit', SSFit:12:4, (n-k-p+1):8, MSFit:12:4, (MSFit/
             MSE):12:4 ) ;
    writeln ;
    writeln( 'Error      ', SSE:12:4, (p-1):8, MSE:12:4 ) ;
    writeln ;
    writeln ;
    writeln( 'Total      ', SSTcorrected:12:4, (n-1):8 ) ;
    if print then
     begin
      writeln( lst ) ;
      writeln( lst, 'A N A L Y S I S   O F   V A R I A N C E' ) ;
      writeln( lst ) ;
      writeln( lst,'  Source         SS         d.f      MS        F') ;
      writeln( lst,'========      =======     ====  ========',
              '  =======' ) ;
      writeln( lst ) ;
      writeln( lst, 'Regression ', SSR:12:4, (k-1):8, MSR:12:4,
               (MSR/MSE):12:4 ) ;
      writeln( lst ) ;
      writeln( lst, 'Lack of fit', SSFit:12:4, (n-k-p+1):8, MSFit:12:4,
               (MSFit/MSE):12:4 ) ;
      writeln( lst ) ;
      writeln( lst, 'Error      ', SSE:12:4, (p-1):8, MSE:12:4 ) ;
      writeln( lst ) ;
      writeln( lst ) ;
      writeln( lst, 'Total      ', SSTcorrected:12:4, (n-1):8 ) ;
      writeln( lst ) ;
      writeln( lst ) ;
     end ;
  pause ;
  end ;
```

```
    if solveXo and continue then
      begin
        clrscr ;
        writeln ;
        writeln( 'Stationary point is located at:' ) ;
        writeln ;
        for i := 2 to f+1 do
          writeln( '    X(', i-1, ') = ', xcrit[i]:12:4 ) ;
        writeln ;
        writeln( 'Dependent variable value at critical point is:' ) ;
        writeln( '    Yo = ', ycrit:12:4 ) ;
        if print then
         begin
          writeln( lst, 'Stationary point is located at:' ) ;
          writeln( lst ) ;
          for i := 2 to f+1 do
            writeln( lst, '    X(', i-1, ') = ', xcrit[i]:12:4 ) ;
          writeln( lst ) ;
          writeln( lst, 'Dependent variable value at critical point is:' ) ;
          writeln( lst, '    Yo = ', ycrit:12:4 ) ;
         end ;
      end ;
end ;  {procedure DisplayResults}

{=========================================================================}

begin              { M A I N   P R O G R A M }
  Initialize ;
  writeln( 'Enter "d" if your data is on disk, any other letter for' ) ;
  write( 'keyboard input ==> ' ) ;
  repeat {$I-} readln( choice ) {$I+} until ioresult = 0 ;
  if upcase( choice ) = 'D'
    then GetData
    else EnterData ;
  FirstCalculations ;
  Orthogonalize ;
  BackSolve ;
  VarianceCovariance ;
  SumOfSquares ;
  StationaryPoint ;
  DisplayResults
end .
```

APPENDIX H

EXPERIMENT INPUT MATRICES

This appendix describes typical design matrices and input files. In each case, a summary of input variable ranges, the specific design matrix, results from AAPMOD runs, and a sample input file for PASCAL program RESPONSE.PAS are provided. The following designs are described:

Section 1: Initial Screening Design

Section 2: Follow-on Screening Design

Section 3: First-Order RSM Design

Section 4: Second-Order RSM Design

Section 5: Follow-on Second-Order RSM Design

INITIAL SCREENING DESIGN

Screening Design -- Resolution III with 7 factors, 3 replications per design point.

		+	-	
A	Pr(arrival)	0.99	0.92	
B	Weapon type	Mk-84	Mk-82	
	equiv. crater radius	40/15 ft	24/9	main runway
		50/20	30/10	aux runway
C	Del error S/D	300/150	100/50	
	REP/DEP	202.5/101.25	67.5/33.75	
D	Number wpns	12	6	
E	Wpn reliability	0.99	0.59	
F	Attack heading	170	140	
G	Weapon spacing	100	50	

Pylons were spaced 20 feet apart. NAREA set on (no overlap search) NSAMP=200, Standardized attacks, ballistic dispersion 30/30 S/D, ballistic dispersion REP/DEP 20.25/20.25 ==> 4000 ft slant range for 5 mils ballistic dispersion

Design matrix:

	A	B	C	D AB	E AC	F BC	G ABC
(1)	-1	-1	-1	1	1	1	-1
a	1	-1	-1	-1	-1	1	1
b	-1	1	-1	-1	1	-1	1
c	-1	-1	1	1	-1	-1	1
ab	1	1	-1	1	-1	-1	-1
ac	1	-1	1	-1	1	-1	-1
bc	-1	1	1	-1	-1	1	-1
abc	1	1	1	1	1	1	1

AAPMOD Results:

Cell	Rep	RW1 Cut	RW2 Cut	Comb Cut	RW1 Crat	RW2 Crat	Comb Crat
1	1	.675	.960	.650	1.665	2.405	1.185
	2	.650	.955	.625	1.780	2.305	1.170
	3	.730	.955	.705	2.030	2.380	1.430
2	1	.300	.440	.140	0.330	0.480	0.140
	2	.280	.415	.140	0.305	0.460	0.140
	3	.290	.470	.110	0.335	0.515	0.110

Cell	Rep	RW1 Cut	RW2 Cut	Comb Cut	RW1 Crat	RW2 Crat	Comb Crat
3	1	.865	.920	.795	1.365	1.605	1.080
	2	.885	.970	.860	1.410	1.735	1.185
	3	.870	.945	.830	1.320	1.620	1.100
4	1	.090	.350	.030	0.090	0.365	0.030
	2	.085	.340	.035	0.085	0.360	0.035
	3	.085	.325	.035	0.085	0.375	0.035
5	1	.915	.955	.875	1.700	1.990	1.320
	2	.915	.915	.840	1.750	1.990	1.330
	3	.925	.940	.870	1.630	2.045	1.235
6	1	.135	.475	.045	0.175	0.475	0.045
	2	.145	.455	.055	0.145	0.730	0.055
	3	.145	.545	.060	0.155	0.855	0.065
7	1	.040	.395	.020	0.040	0.600	0.020
	2	.040	.355	.025	0.055	0.465	0.025
	3	.055	.435	.030	0.070	0.655	0.030
8	1	.460	.975	.450	1.245	1.825	0.735
	2	.495	.975	.480	1.360	1.785	0.740
	3	.505	.965	.490	1.410	1.775	0.760

Sample Input File for RESPONSE.PAS;

Runway 1 probability of cut

```
8
7
0.675        -1  -1  -1   1   1   1   -1
0.300         1  -1  -1  -1  -1   1    1
0.865        -1   1  -1  -1   1  -1    1
0.090        -1  -1   1   1  -1  -1    1
0.915         1   1  -1   1  -1  -1   -1
0.135         1  -1   1  -1   1  -1   -1
0.040        -1   1   1  -1  -1   1   -1
0.460         1   1   1   1   1   1    1
```

Screening Design -- Resolution IV with 7 factors, 3 replications per design point.

		+	−	
A	Pr(arrival)	0.99	0.92	
B	Weapon type	Mk-84	Mk-82	
	equiv. crater radius	40/15 ft	24/9	main runway
		50/20	30/10	aux runway
C	Del error S/D	300/150	100/50	
	REP/DEP	202.5/101.25	67.5/33.75	
D	Number wpns	12	6	
E	Wpn reliability	0.99	0.59	
F	Attack heading	170	140	
G	Weapon spacing	100	50	

Pylons were spaced 20 feet apart. NAREA set on (no overlap search) NSAMP=200, Standardized attacks, ballistic dispersion 30/30 S/D, ballistic dispersion REP/DEP 20.25/20.25 ==> 4000 ft slant range at 5 mils

Design matrix:

	A	B	C	D	AB	AC	AD	BC	BD	CD	E ABC	ABD	G ACD	F BCD	ABCD
(1)	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1	-1	-1	1
a	1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	-1	-1
b	-1	1	-1	-1	-1	1	1	-1	-1	1	1	1	-1	1	-1
c	-1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	1	-1
d	-1	-1	-1	1	1	1	-1	1	-1	-1	-1	1	1	1	-1
ab	1	1	-1	-1	1	-1	-1	-1	-1	1	-1	-1	1	1	1
ac	1	-1	1	-1	-1	1	-1	-1	1	-1	-1	1	-1	1	1
ad	1	-1	-1	1	-1	-1	1	1	-1	-1	1	-1	-1	1	1
bc	-1	1	1	-1	-1	-1	1	1	-1	-1	-1	1	1	-1	1
bd	-1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	-1	1
cd	-1	-1	1	1	1	-1	-1	-1	-1	1	1	1	-1	-1	1
abc	1	1	1	-1	1	1	-1	1	-1	-1	1	-1	-1	-1	-1
abd	1	1	-1	1	1	-1	1	-1	1	-1	-1	1	-1	-1	-1
acd	1	-1	1	1	-1	1	1	-1	-1	1	-1	-1	1	-1	-1
bcd	-1	1	1	1	-1	-1	-1	1	1	1	-1	-1	-1	1	-1
abcd	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Results from AAPMOD runs:

Cell	Rep	RW1 Cut	RW2 Cut	Comb Cut	RW1 Crat	RW2 Crat	Comb Crat
1	1	.250	.605	.155	0.260	0.865	0.155
	2	.190	.610	.105			
	3	.250	.615	.145			
2	1	.755	.760	.550	0.840	0.965	0.555
	2	.725	.825	.605			
	3	.705	.805	.565			
3	1	.540	.935	.515	1.060	2.890	0.950
	2	.485	.945	.450			
	3	.560	.990	.550			
4	1	.085	.560	.050	0.110	0.625	0.050
	2	.055	.515	.030			
	3	.080	.520	.035			
5	1	.765	.410	.325	1.450	0.445	0.340
	2	.775	.400	.330			
	3	.755	.375	.270			
6	1	.525	.735	.355	0.725	0.960	0.390
	2	.545	.645	.335			
	3	.520	.720	.350			
7	1	.015	.240	.000	0.020	0.300	0.000
	2	.010	.275	.000			
	3	.010	.285	.005			
8	1	.735	1.000	.735	2.080	2.705	1.505
	2	.755	.995	.750			
	3	.770	.975	.745			
9	1	.070	.440	.040	0.070	0.525	0.040
	2	.095	.375	.025			
	3	.130	.460	.075			
10	1	.905	.970	.880	1.520	1.725	1.220
	2	.865	.930	.810			
	3	.905	.940	.845			
11	1	.470	.795	.305	0.735	1.545	0.525
	2	.360	.735	.260			
	3	.450	.790	.360			
12	1	.215	.595	.125	0.325	1.215	0.140
	2	.225	.590	.115			
	3	.225	.635	.115			
13	1	.915	.955	.875	1.700	1.990	1.320
	2	.915	.915	.840			
	3	.925	.940	.870			
14	1	.165	.355	.065	0.170	0.355	0.065
	2	.155	.370	.055			
	3	.145	.360	.055			
15	1	.205	.700	.150	0.325	1.210	0.185
	2	.205	.705	.155			
	3	.150	.710	.095			
16	1	.460	.975	.450	1.245	1.825	0.735
	2	.495	.975	.480			
	3	.505	.965	.490			

Sample Input File for RESPONSE.PAS:

```
                    Runway 1 probability of cut

 16
 15
 0.250    -1  -1  -1  -1   1   1   1   1   1   1  -1  -1  -1  -1   1
 0.755     1  -1  -1  -1  -1  -1  -1   1   1   1   1   1   1  -1  -1
 0.540    -1   1  -1  -1  -1   1   1  -1  -1   1   1   1  -1   1  -1
 0.085    -1  -1   1  -1   1  -1   1  -1   1  -1   1  -1   1   1  -1
 0.765    -1  -1  -1   1   1   1  -1   1  -1  -1  -1   1   1   1  -1
 0.525     1   1  -1  -1   1  -1  -1  -1  -1   1  -1  -1   1   1   1
 0.015     1  -1   1  -1  -1   1  -1  -1   1  -1  -1   1  -1   1   1
 0.735     1  -1  -1   1  -1  -1   1   1  -1  -1   1  -1  -1   1   1
 0.070    -1   1   1  -1  -1  -1   1   1  -1  -1  -1   1   1  -1   1
 0.905    -1   1  -1   1  -1   1  -1  -1   1  -1   1  -1   1  -1   1
 0.470    -1  -1   1   1   1  -1  -1  -1  -1   1   1   1  -1  -1   1
 0.215     1   1   1  -1   1   1  -1   1  -1  -1   1  -1  -1  -1  -1
 0.915     1   1  -1   1   1  -1   1  -1   1  -1  -1   1  -1  -1  -1
 0.165     1  -1   1   1  -1   1   1  -1  -1   1  -1  -1   1  -1  -1
 0.205    -1   1   1   1  -1  -1  -1   1   1   1  -1  -1  -1   1  -1
 0.460     1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
```

FOLLOW-ON SCREENING DESIGN

Screening Design -- Resolution IV with 7 factors, 10 replications per design point

		+	-	
A	Pr(arrival)	0.99	0.92	
B	Weapon type	MK-84	MK-82	
	equiv. crater radius	40/15 ft	24/9	main runway
		50/20	30/10	aux runway
C	Del error S/D	300/150	100/50	
	REP/DEP	202.5/101.25	67.5/33.75	
D	Number wpns	12	6	
E	Wpn reliability	0.99	0.59	
F	Attack heading	170	140	
G	Weapon spacing	100	50	

Pylons were spaced 20 feet apart. NAREA set on (no overlap search) NSAMP=200, Standardized attacks, ballistic dispersion 30/30 S/D, ballistic dispersion REP/DEP 20.25/20.25 ==> 4000 ft slant range for 5 mils ballistic dispersion

Design matrix:

Same as previous Resolution IV design.

AAPMOD Output:

Each design point was run with 10 replications. Probabilities of cut for combined runways are depicted.

	DESIGN POINT							
Replication	1	2	3	4	5	6	7	8
1	0.300	0.550	0.650	0.150	0.350	0.450	0.000	0.750
2	0.100	0.700	0.500	0.000	0.400	0.250	0.000	0.700
3	0.150	0.550	0.650	0.000	0.500	0.200	0.000	0.650
4	0.100	0.550	0.550	0.000	0.250	0.450	0.000	0.900
5	0.100	0.650	0.400	0.000	0.250	0.350	0.000	0.800
6	0.200	0.550	0.500	0.100	0.200	0.400	0.000	0.850
7	0.100	0.400	0.350	0.100	0.600	0.150	0.000	0.700
8	0.150	0.450	0.650	0.100	0.200	0.350	0.000	0.550
9	0.250	0.500	0.450	0.050	0.500	0.350	0.000	0.750
10	0.100	0.600	0.450	0.000	0.350	0.500	0.000	0.700

Replication	DESIGN POINT 1	2	3	4	5	6	7	8
1	0.050	0.950	0.400	0.150	1.000	0.050	0.100	0.450
2	0.000	0.900	0.500	0.150	0.850	0.000	0.150	0.400
3	0.000	0.800	0.550	0.150	0.850	0.050	0.200	0.500
4	0.050	0.800	0.200	0.050	0.850	0.100	0.100	0.650
5	0.050	0.900	0.250	0.100	0.750	0.050	0.100	0.400
6	0.100	0.800	0.300	0.050	0.950	0.150	0.150	0.450
7	0.050	0.850	0.450	0.300	0.850	0.050	0.100	0.400
8	0.050	0.950	0.300	0.100	1.000	0.050	0.150	0.300
9	0.050	0.900	0.400	0.150	0.900	0.100	0.250	0.500
10	0.000	0.950	0.500	0.050	0.750	0.050	0.200	0.450

Input file for RESPONSE.PAS:

The file required one-time modifications to RESPONSE.PAS due to the large number of data points. The input file was similar to the ones shown previously. In this case, each design point was replicated 10 times with appropriate system responses inserted as before.

FIRST-ORDER RSM DESIGN

First-Order Response Surface Methodology -- variables to fit were weapon type, weapon reliability, number of weapons and delivery error.

		+	-	
A	Pr(arrival)	0.955	0.955	
B	Weapon type	Mk-84	Mk-82	
	equiv. crater radius	40/15 ft	24/9	main runway
		50/20	30/10	aux runway
C	Del error S/D	300/150	100/50	
	REP/DEP	202.5/101.25	67.5/33.75	
D	Number wpns	12	6	
E	Wpn reliability	0.99	0.59	
F	Attack heading	155	155	
G	Weapon spacing	75	75	

Pylons were spaced 20 feet apart. NAREA set on (no overlap search) NSAMP=200, Standardized attacks, ballistic dispersion 30/30 S/D, ballistic dispersion REP/DEP 20.25/20.25 ==> 4000 ft slant range for 5 mils ballistic dispersion

Fractional factorial design matrix (half fraction):

B	C	D	E
-1	-1	-1	-1
1	1	-1	-1
-1	-1	1	1
1	-1	1	-1
1	-1	-1	1
-1	1	1	-1
-1	1	-1	1
1	1	1	1

AAPMOD Output:

Cell	RW1 Cut	RW2 Cut	Comb Cut	RW1 Crat	RW2 Crat	Comb Crat
1	.340	.550	.215	0.355	0.620	0.215
2	.100	.465	.065	0.021	0.035	0.017
3	.980	.910	.895	2.550	1.415	1.310
4	.905	.865	.780	1.685	1.290	0.950
5	.875	.990	.865	1.925	2.225	1.500
6	.275	.445	.135	0.305	0.525	0.140
7	.195	.530	.085	0.250	0.635	0.090
8	.740	.950	.700	1.600	2.010	1.175

Sample Input File for RESPONSE.PAS:

```
Runway 1 probability of cut

 8
 4
 0.340   -1   -1   -1   -1
 0.100    1    1   -1   -1
 0.980   -1   -1    1    1
 0.905    1   -1    1   -1
 0.875    1   -1   -1    1
 0.275   -1    1    1   -1
 0.195   -1    1   -1    1
 0.740    1    1    1    1
```

First-Order Response Methodology fitting delivery error, weapon reliability, attack heading, and weapon spacing.

Crater radii were updated for this and subsequent analyses. The new figures are displayed below. Weapon type and number of weapons were fixed for this analysis, but several combinations were of interest. The entire first-order RSM analysis was run for 12 Mk-82 bombs and then for 6 Mk-84 bombs.

		+	−	
A	Pr(arrival)	0.95	0.95	
B	Weapon type	Mk-84 --OR--	Mk-82	analyzed separately
	equiv. crater radius	28.8/13.8	17.6/7.6	main runway
		30/15	20/10	aux runway
C	Del error S/D	300/150	100/50	
D	Number wpns	12 --OR--	6	analyzed separately
E	Wpn reliability	0.99	0.59	
F	Attack heading	170	140	
G	Weapon spacing	100	50	

Pylons were spaced 20 feet apart. NAREA set on (no overlap search) NSAMP=200, Standardized attacks, ballistic dispersion 30/30 S/D, ballistic dispersion REP/DEP 20.25/20.25 ==) 4000 ft slant range for 5 mils ballistic dispersion

Fractional factorial design matrix (half fraction):

C	E	F	G
-1	-1	-1	-1
1	1	-1	-1
-1	-1	1	1
1	-1	1	-1
1	-1	-1	1
-1	1	1	-1
-1	1	-1	1
1	1	1	1

new figures are displayed below. Weapon type and number of weapons were fixed for this analysis, but several combinations were of interest. The entire first-order RSM analysis was run for 12 Mk-82 bombs and then for 6 Mk-84 bombs.

		+	-	
A	Pr(arrival)	0.95	0.95	
B	Weapon type	Mk-84 --OR--	Mk-82	analyzed separately
	equiv. crater radius	28.8/13.8	17.6/7.6	main runway
		30/15	20/10	aux runway
C	Del error S/D	300/150	100/50	
D	Number wpns	12 --OR--	6	analyzed separately
E	Wpn reliability	0.99	0.59	
F	Attack heading	170	140	
G	Weapon spacing	100	50	

Pylons were spaced 20 feet apart. NAREA set on (no overlap search)
NSAMP=200, Standardized attacks, ballistic dispersion 30/30 S/D, ballistic dispersion REP/DEP 20.25/20.25 ==> 4000 ft slant range for 5 mils ballistic dispersion

Fractional factorial design matrix (half fraction):

C	E	F	G
-1	-1	-1	-1
1	1	-1	-1
-1	-1	1	1
1	-1	1	-1
1	-1	-1	1
-1	1	1	-1
-1	1	-1	1
1	1	1	1

AAPMOD Output (MK-82):

Cell	RW1 Cut	RW2 Cut	Comb Cut	RW1 Crat	RW2 Crat	Comb Crat
1	.520	.615	.335	0.585	0.720	0.340
2	.360	.665	.240	0.445	0.950	0.260
3	.710	.270	.210	1.145	0.275	0.210
4	.075	.450	.035	0.080	0.530	0.035
5	.045	.160	.005	0.045	0.160	0.005
6	.640	.940	.605	1.375	1.805	0.895
7	.520	.630	.330	0.520	0.665	0.330
8	.335	.565	.190	0.590	0.615	0.205

AAPMOD Output (MK-84):

Cell	RW1 Cut	RW2 Cut	Comb Cut	RW1 Crat	RW2 Crat	Comb Crat
1	.360	.645	.220	0.380	0.885	0.225
2	.180	.455	.075	0.205	0.660	0.075
3	.385	.380	.150	0.445	0.400	0.150
4	.040	.265	.015	0.040	0.320	0.015
5	.050	.225	.020	0.055	0.225	0.020
6	.555	.905	.500	0.905	1.820	0.695
7	.795	.705	.545	0.995	0.835	0.565
8	.195	.545	.090	0.235	0.645	0.095

Sample Input file for RESPONSE.PAS:

```
Runway 1 probability of cut

8
4
0.520   -1   -1   -1   -1
0.360    1    1   -1   -1
0.710   -1   -1    1    1
0.075    1   -1    1   -1
0.045    1   -1   -1    1
0.640   -1    1    1   -1
0.520   -1    1   -1    1
0.335    1    1    1    1
```

SECOND-ORDER RSM DESIGN

Initial Second-Order RSM

		-2	-1	0	+1	+2
A	Pr(arrival)	0.95	0.95	0.95	0.95	0.95
B	Weapon type	Mk-84 --OR-- Mk-82 analyzed separately				
	equiv. crater radius	28.8/13.8	17.6/7.6	main runway		
		30/15	20/10	aux runway		
C	Del error S/D	0/0	150/75	300/150	450/225	600/300
D	Number wpns	12 --OR-- 6 analyzed separately				
E	Wpn reliability	0.59	0.69	0.79	0.89	0.99
F	Attack heading	90.0	112.5	135.0	157.5	180.0
G	Weapon spacing	20	40	60	80	100

Pylons were spaced 20 feet apart. NAREA set on (no overlap search) NSAMP=200, Standardized attacks, ballistic dispersion 30/30 S/D, ballistic dispersion REP/DEP 20.25/20.25 ==> 4000 ft slant range for 5 mils ballistic dispersion

Central Composite Design for 4 variables and 7 center points:

C	E	F	G	CC	EE	FF	GG	CE	CF	CG	EF	EG	FG
-1	-1	-1	-1	1	1	1	1	1	1	1	1	1	1
1	-1	-1	-1	1	1	1	1	-1	-1	-1	1	1	1
-1	1	-1	-1	1	1	1	1	-1	1	1	-1	-1	1
1	1	-1	-1	1	1	1	1	1	-1	-1	-1	-1	1
-1	-1	1	-1	1	1	1	1	1	-1	1	-1	1	-1
1	-1	1	-1	1	1	1	1	-1	1	-1	-1	1	-1
-1	1	1	-1	1	1	1	1	-1	-1	1	1	-1	-1
1	1	1	-1	1	1	1	1	1	1	-1	1	-1	-1
-1	-1	-1	1	1	1	1	1	1	1	-1	1	-1	-1
1	-1	-1	1	1	1	1	1	-1	-1	1	1	-1	-1
-1	1	-1	1	1	1	1	1	-1	1	-1	-1	1	-1
1	1	-1	1	1	1	1	1	1	-1	1	-1	1	-1
-1	-1	1	1	1	1	1	1	1	-1	-1	-1	-1	1
1	-1	1	1	1	1	1	1	-1	1	1	-1	-1	1
-1	1	1	1	1	1	1	1	-1	-1	-1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1
-2	0	0	0	4	0	0	0	0	0	0	0	0	0
2	0	0	0	4	0	0	0	0	0	0	0	0	0
0	-2	0	0	0	4	0	0	0	0	0	0	0	0
0	2	0	0	0	4	0	0	0	0	0	0	0	0
0	0	-2	0	0	0	4	0	0	0	0	0	0	0
0	0	2	0	0	0	4	0	0	0	0	0	0	0
0	0	0	-2	0	0	0	4	0	0	0	0	0	0
0	0	0	2	0	0	0	4	0	0	0	0	0	0

C	E	F	G	CC	EE	FF	GG	CE	CF	CG	EF	EG	FG
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

AAPMOD Output (Mk-82):

Cell	RW1 Cut	RW2 Cut	Comb Cut	RW1 Crat	RW2 Crat	Comb Crat
1	.430	.775	.345	0.480	1.460	0.360
2	.045	.360	.020	0.045	0.480	0.020
3	.595	.905	.535	0.785	2.115	0.650
4	.140	.520	.065	0.140	0.875	0.065
5	.530	.765	.405	0.785	1.250	0.470
6	.090	.365	.025	0.110	0.415	0.025
7	.615	.835	.520	1.300	1.625	0.810
8	.120	.385	.040	0.160	0.555	0.040
9	.070	.645	.040	0.070	0.760	0.040
10	.040	.370	.010	0.040	0.460	0.010
11	.270	.765	.225	0.275	1.035	0.230
12	.060	.450	.025	0.060	0.580	0.025
13	.580	.460	.265	0.720	0.495	0.265
14	.075	.300	.015	0.085	0.315	0.015
15	.825	.685	.580	1.310	0.745	0.600
16	.235	.395	.095	0.280	0.455	0.095
17	.345	.525	.185	0.355	0.620	0.185
18	.045	.280	.010	0.045	0.315	0.010
19	.100	.345	.020	0.100	0.380	0.020
20	.395	.640	.275	0.450	0.880	0.290
21	.140	.700	.080	0.145	1.350	0.080
22	.055	.605	.025	0.085	0.825	0.035
23	.120	.425	.040	0.170	0.850	0.050
24	.150	.410	.075	0.150	0.425	0.075
25	.225	.570	.110	0.250	0.705	0.115
26	.235	.460	.110	0.245	0.560	0.110
27	.280	.550	.155	0.320	0.635	0.155
28	.215	.540	.135	0.235	0.645	0.135
29	.225	.555	.125	0.240	0.690	0.130
30	.205	.500	.105	0.215	0.645	0.105
31	.260	.520	.140	0.275	0.675	0.145

Sample Input File for RESPONSE.PAS:

```
31
14
0.430  -1  -1  -1  -1   1   1   1   1   1   1   1   1   1   1
0.045   1  -1  -1  -1   1   1   1   1  -1  -1  -1   1   1   1
0.595  -1   1  -1  -1   1   1   1   1  -1   1   1  -1  -1   1
0.140   1   1  -1  -1   1   1   1   1   1  -1  -1  -1  -1   1
0.530  -1  -1   1  -1   1   1   1   1   1  -1   1  -1   1  -1
0.090   1  -1   1  -1   1   1   1   1  -1   1  -1  -1   1  -1
0.615  -1   1   1  -1   1   1   1   1  -1  -1   1   1  -1  -1
0.120   1   1   1  -1   1   1   1   1   1   1  -1   1  -1  -1
0.070  -1  -1  -1   1   1   1   1   1   1   1  -1   1  -1  -1
0.040   1  -1  -1   1   1   1   1   1  -1  -1   1   1  -1  -1
0.270  -1   1  -1   1   1   1   1   1  -1   1  -1  -1   1  -1
0.060   1   1  -1   1   1   1   1   1   1  -1   1  -1   1  -1
0.580  -1  -1   1   1   1   1   1   1   1  -1  -1  -1  -1   1
0.075   1  -1   1   1   1   1   1   1  -1   1   1  -1  -1   1
0.825  -1   1   1   1   1   1   1   1  -1  -1  -1   1   1   1
0.235   1   1   1   1   1   1   1   1   1   1   1   1   1   1
0.345  -2   0   0   0   4   0   0   0   0   0   0   0   0   0
0.045   2   0   0   0   4   0   0   0   0   0   0   0   0   0
0.100   0  -2   0   0   0   4   0   0   0   0   0   0   0   0
0.395   0   2   0   0   0   4   0   0   0   0   0   0   0   0
0.140   0   0  -2   0   0   0   4   0   0   0   0   0   0   0
0.055   0   0   2   0   0   0   4   0   0   0   0   0   0   0
0.120   0   0   0  -2   0   0   0   4   0   0   0   0   0   0
0.150   0   0   0   2   0   0   0   4   0   0   0   0   0   0
0.225   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0.235   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0.280   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0.215   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0.225   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0.205   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0.260   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

FOLLOW-ON SECOND-ORDER RSM DESIGN

Follow-on Second-Order RSM

		-2	-1	0	+1	+2
A	Pr(arrival)	0.95	0.95	0.95	0.95	0.95
B	Weapon type	Mk-84 --OR-- Mk-82 analyzed separately				
	equiv. crater radius	28.8/13.8	17.6/7.6	main runway		
		30/15	20/10	aux runway		
C	Del error S/D	0/0	75/37.5	150/75	225/112.5	300/150
D	Number wpns	12 --OR-- 6 analyzed separately				
E	Wpn reliability	0.79	0.84	0.89	0.94	0.99
F	Attack heading	135.0	146.3	157.5	168.7	180.0
G	Weapon spacing	60	70	80	90	100

Pylons were spaced 20 feet apart. NAREA set off (overlap search) NSAMP=200, Standardized attacks, ballistic dispersion 30/30 S/D, ballistic dispersion REP/DEP 20.25/20.25 ==> 4000 ft slant range for 5 mils ballistic dispersion

Design Matrix: same as initial second-order RSM

AAPMOD Output (MK-82) including expected area to fill:

Cell	RW1 Cut	RW2 Cut	Comb Cut	RW1 Crat	RW2 Crat	Comb Crat	RW1 Fill	RW2 Fill	Comb Fill
1	0.715	0.630	0.430	0.945	0.785	0.460	340	317	151
2	0.530	0.600	0.335	0.595	0.675	0.350	214	248	128
3	0.845	0.710	0.600	1.275	0.850	0.655	492	347	227
4	0.595	0.650	0.375	0.775	0.810	0.395	294	331	134
5	0.875	0.720	0.630	2.120	0.870	0.730	856	370	297
6	0.360	0.665	0.270	0.535	0.775	0.295	202	293	102
7	0.925	0.755	0.700	2.470	0.915	0.815	1040	365	306
8	0.385	0.680	0.270	0.635	0.875	0.310	241	356	107
9	0.485	0.515	0.255	0.520	0.575	0.255	183	224	86
10	0.400	0.475	0.190	0.430	0.520	0.190	141	191	65
11	0.645	0.530	0.355	0.755	0.560	0.360	276	209	119
12	0.530	0.575	0.320	0.590	0.625	0.325	209	237	106
13	0.915	0.540	0.510	2.230	0.590	0.545	945	229	205
14	0.445	0.485	0.210	0.650	0.505	0.210	229	117	72
15	0.960	0.605	0.590	2.785	0.665	0.625	1194	250	216
16	0.510	0.635	0.340	0.885	0.720	0.355	348	259	124
17	0.795	0.555	0.435	1.220	0.645	0.470	470	232	159
18	0.465	0.555	0.290	0.625	0.665	0.300	237	245	106
19	0.750	0.595	0.445	1.055	0.640	0.455	401	229	142
20	0.875	0.725	0.630	1.560	0.890	0.705	648	362	264

Cell	RW1 Cut	RW2 Cut	Comb Cut	RW1 Crat	RW2 Crat	Comb Crat	RW1 Fill	RW2 Fill	Comb Fill
21	0.475	0.665	0.330	0.515	0.765	0.335	188	299	117
22	0.255	0.700	0.165	0.395	0.885	0.185	151	383	65
23	0.825	0.775	0.645	1.480	1.065	0.785	578	456	276
24	0.770	0.515	0.405	1.030	0.540	0.410	390	207	140
25	0.825	0.685	0.580	1.310	0.745	0.600	501	286	212
26	0.800	0.635	0.500	1.300	0.755	0.565	515	286	212
27	0.820	0.665	0.550	1.300	0.790	0.615	538	334	233
28	0.790	0.640	0.495	1.395	0.715	0.530	518	264	185
29	0.790	0.605	0.490	1.245	0.665	0.520	483	255	187
30	0.855	0.650	0.570	1.390	0.745	0.615	541	267	207
31	0.825	0.630	0.520	1.385	0.735	0.560	544	280	207

Sample Input File for RESPONSE.PAS:

```
31
14
0.715  -1  -1  -1  -1   1   1   1   1   1   1   1   1   1   1
0.530   1  -1  -1  -1   1   1   1   1  -1  -1  -1   1   1   1
0.845  -1   1  -1  -1   1   1   1   1  -1   1   1  -1  -1   1
0.595   1   1  -1  -1   1   1   1   1   1  -1  -1  -1  -1   1
0.875  -1  -1   1  -1   1   1   1   1   1  -1   1  -1   1  -1
0.360   1  -1   1  -1   1   1   1   1  -1   1  -1  -1   1  -1
0.925  -1   1   1  -1   1   1   1   1  -1  -1   1   1  -1  -1
0.385   1   1   1  -1   1   1   1   1   1   1  -1   1  -1  -1
0.485  -1  -1  -1   1   1   1   1   1   1   1  -1   1  -1  -1
0.400   1  -1  -1   1   1   1   1   1  -1  -1   1   1  -1  -1
0.645  -1   1  -1   1   1   1   1   1  -1   1  -1  -1   1  -1
0.530   1   1  -1   1   1   1   1   1   1  -1   1  -1   1  -1
0.915  -1  -1   1   1   1   1   1   1   1  -1  -1  -1  -1   1
0.445   1  -1   1   1   1   1   1   1  -1   1   1  -1  -1   1
0.960  -1   1   1   1   1   1   1   1  -1  -1  -1   1   1   1
0.510   1   1   1   1   1   1   1   1   1   1   1   1   1   1
0.795  -2   0   0   0   4   0   0   0   0   0   0   0   0   0
0.465   2   0   0   0   4   0   0   0   0   0   0   0   0   0
0.750   0  -2   0   0   0   4   0   0   0   0   0   0   0   0
0.875   0   2   0   0   0   4   0   0   0   0   0   0   0   0
0.475   0   0  -2   0   0   0   4   0   0   0   0   0   0   0
0.255   0   0   2   0   0   0   4   0   0   0   0   0   0   0
0.825   0   0   0  -2   0   0   0   4   0   0   0   0   0   0
0.770   0   0   0   2   0   0   0   4   0   0   0   0   0   0
0.825   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0.800   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0.820   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0.790   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0.790   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0.855   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0.825   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

APPENDIX J

SELECTED EXPERIMENT RESULTS

This appendix contains selected output from the response surface methodology program located at Appendix G. The output consists of regression (sometimes known as beta) coefficients which define the regression equations which are to explain the input data. For second-order response surface methodology, analysis of variance (ANOVA) and stationary point data are included. These are explained in Chapter VIII of the thesis.

The exact form of the regression equation for a particular analysis can vary, but the resolution III first-order regression equations take a form similar to the following:

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + b_4x_4 + b_5x_5 + b_6x_6 + b_7x_7$$

where y is the predicted system response (probability of cut, etc.)

b_i is a regression coefficient for $i = 0, \ldots, 7$

x_i is a coded independent variable for $i = 1, \ldots, 7$

Screening Design -- Resolution III with 7 Factors:

Regression Coefficient	RW1 Cut	RW2 Cut	Comb Cut	RW1 Crat	RW2 Crat	Comb Crat
b(0)	0.435	0.684	0.376	0.826	1.218	0.569
b(1)	0.018	0.028	0.002	0.036	-0.026	-0.009
b(2)	0.135	0.128	0.159	0.261	0.287	0.219
b(3)	-0.254	-0.135	-0.239	-0.439	-0.402	-0.362
b(4)	0.100	0.126	0.126	0.349	0.428	0.248
b(5)	0.099	0.149	0.109	0.286	0.359	0.192
b(6)	-0.066	0.009	-0.061	-0.006	0.109	-0.049
b(7)	-0.006	-0.013	-0.022	-0.069	-0.149	-0.073

Screening Design -- Resolution IV with 7 Factors

Regression Coefficient	RW1 Cut	RW2 Cut	Comb Cut	RW1 Crat	RW2 Crat	Comb Crat
b(0)	0.442	0.627	0.353	0.790	1.259	0.510
b(1)	0.031	-0.050	0.041	0.098	0.030	0.078
b(2)	0.037	0.161	0.070	0.082	0.283	0.112
b(3)	-0.231	-0.044	-0.195	-0.415	-0.309	-0.293
b(4)	0.135	0.018	0.130	0.363	0.216	0.226
b(5)	0.018	0.077	-0.013	0.029	-0.075	-0.054
b(6)	-0.028	0.009	-0.039	-0.033	-0.057	-0.060
b(7)	-0.040	-0.024	0.007	0.047	0.213	0.092
b(8)	-0.010	-0.066	-0.037	0.035	-0.040	-0.054
b(9)	0.007	0.094	0.035	-0.037	-0.070	0.017
b(10)	-0.021	0.106	-0.025	-0.120	0.068	-0.066
b(11)	0.078	0.072	0.108	0.200	0.428	0.199
b(12)	0.057	0.062	0.039	0.100	0.052	0.047
b(13)	0.024	0.024	-0.014	-0.023	-0.331	-0.087
b(14)	-0.026	-0.058	-0.031	0.087	0.111	0.008
b(15)	-0.013	-0.032	0.022	0.042	0.047	0.060

Screening Design -- Resolution IV, 7 Factors with 10 Replications for Each of 16 Design Points. Due to the large amount of data, combined probability of cut for runways #1 and #2 is evaluated.

Regression Coefficient	Comb Cut
b(0)	0.355
b(1)	0.038
b(2)	0.068
b(3)	-0.197
b(4)	0.133
b(5)	-0.012
b(6)	-0.036
b(7)	0.006
b(8)	-0.034
b(9)	0.034
b(10)	-0.028
b(11)	0.106
b(12)	0.042
b(13)	-0.013
b(14)	-0.029
b(15)	0.019

First-Order Response Surface Methodology

The following regression coefficients fit weapons type, number of weapons, weapon reliability, and delivery error to system responses.

Regression Coefficient	RW1 Cut	RW2 Cut	Comb Cut	RW1 Crat	RW2 Crat	Comb Crat
b(0)	0.551	0.713	0.468	0.109	1.094	0.675
b(1)	0.104	0.104	0.135	0.221	0.296	0.236
b(2)	-0.223	-0.116	-0.221	-0.542	-0.293	-0.319
b(3)	0.174	0.079	0.160	0.449	0.216	0.219
b(4)	0.146	0.132	0.169	0.495	0.477	0.344

First-Order Response Surface Methodology

The following regression coefficients fit delivery error, weapon reliability, attack heading, and weapon spacing to system responses.

MK-82 Analysis

Regression Coefficient	RW1 Cut	RW2 Cut	Comb Cut	RW1 Crat	RW2 Crat	Comb Crat
b(0)	0.400	0.537	0.244	0.598	0.715	0.285
b(1)	-0.197	-0.769	-0.126	-0.308	-0.151	-0.159
b(2)	0.063	0.163	0.098	0.134	0.294	0.130
b(3)	0.039	0.019	0.016	0.199	0.091	0.051
b(4)	0.002	-0.131	-0.060	-0.023	-0.206	-0.098

MK-84 Analysis

Regression Coefficient	RW1 Cut	RW2 Cut	Comb Cut	RW1 Crat	RW2 Crat	Comb Crat
b(0)	0.320	0.516	0.202	0.408	0.724	0.230
b(1)	-0.203	-0.143	-0.152	-0.274	-0.261	-0.179
b(2)	0.111	0.137	0.101	0.178	0.266	0.128
b(3)	-0.026	0.008	-0.013	-0.001	0.073	0.009
b(4)	0.004	-0.052	-0.001	0.025	-0.198	-0.023

Second-Order Response Surface Methodology

The following regression coefficients fit delivery error, weapon reliability, attack heading, and weapon spacing to system responses. The same four independent variables will be used for the remainder of the analysis.

MK-82 Results

Regression Coefficient	RW1 Cut	RW2 Cut	Comb Cut	RW1 Crat	RW2 Crat	Comb Crat
b(0)	0.235	0.528	0.126	0.254	0.651	0.128
b(1)	-0.155	-0.133	-0.124	-0.226	-0.248	-0.145
b(2)	0.066	0.062	0.061	0.111	0.140	0.077
b(3)	0.052	-0.033	0.024	0.114	-0.123	0.035
b(4)	-0.015	0.036	-0.026	-0.042	-0.199	-0.046
b(5)	0.011	-0.020	0.012	0.024	-0.022	0.016
b(6)	0.024	0.003	0.024	0.043	0.019	0.030
b(7)	-0.013	0.043	0.001	0.003	0.133	0.006
b(8)	-0.004	-0.016	0.002	0.014	0.021	0.007
b(9)	-0.024	-0.012	-0.041	-0.078	-0.048	-0.063
b(10)	-0.059	0.006	-0.036	-0.135	0.038	-0.051
b(11)	0.028	0.038	0.043	0.062	0.181	0.072
b(12)	0.003	-0.005	0.006	0.045	-0.034	0.014
b(13)	0.016	0.009	0.014	0.003	-0.049	-0.004
b(14)	0.071	-0.011	0.039	0.065	0.016	0.026

The following analyses of variance use the center point variance to calculate statistics for lack of fit and regression. This was later modified such that lack of fit was compared to center point error variance, but regression sum of squares was compared to the combination of center point variance and lack of fit. The F-values change, but the p-value remains approximately the same due to the increased number of degrees of freedom. This appendix contains results using the older method.

Runway 1 -- Probability of Cut

ANALYSIS OF VARIANCE

Source	Sum of Squares	d.f.	Mean Squares	F	p
Regression	0.9388	14	0.0671	96.953	0.0000
Lack of Fit	0.2774	10	0.0277	40.100	0.0001
Error	0.0042	6	0.0007		
Total	1.2203	30			

STATIONARY POINT ANALYSIS

The stationary point is located at:

x(1) = 2.9262
x(2) = -0.3345
x(3) = -0.6945
x(4) = 1.4739

with a probability of cut of:

y = -0.0312

Runway 2 -- Probability of Cut

ANALYSIS OF VARIANCE

Source	Sum of Squares	d.f.	Mean Squares	F	p
Regression	0.6803	14	0.0486	33.928	0.0002
Lack of Fit	0.1142	10	0.0114	7.977	0.0097
Error	0.0086	6	0.0014		
Total	0.8031	30			

STATIONARY POINT ANALYSIS

The stationary point is located at:

x(1) = 32.6031
x(2) = 2.5116
x(3) = 3.3042
x(4) = 37.4168

with a probability of cut of:

y = -2.2867

Combined Runways -- Probability of Cut

ANALYSIS OF VARIANCE

Source	Sum of Squares	d.f.	Mean Squares	F	p
Regression	0.6120	14	0.0437	126.630	0.0000
Lack of Fit	0.1953	10	0.0195	56.572	0.0000
Error	0.0021	6	0.0003		
Total	0.8094	30			

STATIONARY POINT ANALYSIS

The stationary point is located at:

x(1) = 1.6829
x(2) = 0.0144
x(3) = -1.2578
x(4) = 0.9393

with a probability of cut of:

y = -0.0052

Runway 1 -- Expected Number of Craters to Fill

ANALYSIS OF VARIANCE

Source	Sum of Squares	d.f.	Mean Squares	F	p
Regression	2.4956	14	0.1783	153.420	0.0000
Lack of Fit	0.8176	10	0.0818	70.369	0.0000
Error	0.0070	6	0.0012		
Total	3.3202	30			

STATIONARY POINT ANALYSIS

The stationary point is located at:

x(1) = 1.3871
x(2) = 0.5035
x(3) = -1.0714
x(4) = 0.8697

with an expected number of craters to fill:

y = 0.0463

Runway 2 -- Expected Number of Craters to Fill

ANALYSIS OF VARIANCE

Source	Sum of Squares	d.f.	Mean Squares	F	p
Regression	4.4604	14	0.3186	140.339	0.0000
Lack of Fit	0.5966	10	0.0597	26.280	0.0004
Error	0.0136	6	0.0023		
Total	5.0707	30			

STATIONARY POINT ANALYSIS

The stationary point is located at:

x(1) = 0.3539
x(2) = -1.9384
x(3) = 0.1110
x(4) = 0.9233

with an expected number of craters to fill:

y = 0.3727

Combined Runways -- Expected Number of Craters to Fill

ANALYSIS OF VARIANCE

Source	Sum of Squares	d.f.	Mean Squares	F	p
Regression	0.9585	14	0.0685	196.280	0.0000
Lack of Fit	0.2944	10	0.0294	84.400	0.0000
Error	0.0021	6	0.0003		
Total	1.2550	30			

STATIONARY POINT ANALYSIS

The stationary point is located at:

x(1) = 0.0624
x(2) = -0.0734
x(3) = -1.0514
x(4) = 0.8377

with an expected number of craters to fill:

y = 0.0250

Second-Order Response Surface Methodology -- Reduced Surface

The following regression coefficients fit delivery error, weapon reliability, attack heading, and weapon spacing to system responses.

MK-82 Results

Coeff	RW1 Cut	RW2 Cut	Comb Cut	RW1 Crat	RW2 Crat	Comb Crat	RW1 Fill	RW2 Fill	Comb Fill
b(0)	.815	.644	.529	1.319	.736	.572	520	282	206
b(1)	-.136	-.010	-.085	-.383	-.011	-.098	-163	-11	-36
b(2)	.038	.032	.045	.131	.051	.054	62	26	20
b(3)	.008	.020	.014	.258	.031	.025	118	13	13
b(4)	-.019	-.065	-.055	-.059	-.119	-.079	-22	-58	-30
b(5)	-.049	-.026	-.045	-.074	-.029	-.051	-30	-15	-19
b(6)	-.004	-.000	-.001	.022	-.002	-.002	12	-2	-2
b(7)	-.116	.005	-.074	-.191	.013	-.082	-76	10	-30
b(8)	-.007	-.004	-.004	.009	.008	.002	2	7	-1
b(9)	-.006	.008	-.008	-.050	.024	-.008	-23	19	-2
b(10)	-.084	-.004	-.058	-.362	-.002	-.067	-161	-6	-29
b(11)	.023	.013	.029	.033	.017	.038	6	4	16
b(12)	-.019	.001	-.010	.021	.009	-.010	12	5	-5
b(13)	.008	.009	.010	.014	.002	.008	5	1	3
b(14)	.057	-.004	.025	.130	-.007	.020	57	-9	4

<u>Runway 1 -- Probability of Cut</u>

A N A L Y S I S O F V A R I A N C E

Source	Sum of Squares	d.f.	Mean Squares	F	p
Regression	1.0985	14	0.0785	142.662	0.0000
Lack of Fit	0.0067	10	0.0007	15.766	0.2015
Error	0.0033	6	0.0006		
Total	1.1085	30			

S T A T I O N A R Y P O I N T A N A L Y S I S

The stationary point is located at:

x(1) = -1.9742
x(2) = 6.2768
x(3) = 0.3039
x(4) = 0.2587

with a probability of cut of:

y = 1.0686

Runway 2 -- Probability of Cut

ANALYSIS OF VARIANCE

Source	Sum of Squares	d.f.	Mean Squares	F	p
Regression	0.1666	14	0.0119	17.980	0.0010
Lack of Fit	0.0110	10	0.0011	1.664	0.2753
Error	0.0040	6	0.0007		
Total	0.1816	30			

STATIONARY POINT ANALYSIS

The stationary point is located at:

x(1) = -0.0928
x(2) = 3.4624
x(3) = -3.2109
x(4) = -2.7520

with a probability of cut of:

y = 0.7588

Combined Runways -- Probability of Cut

ANALYSIS OF VARIANCE

Source	Sum of Squares	d.f.	Mean Squares	F	p
Regression	0.5812	14	0.0415	29.931	0.0002
Lack of Fit	0.0739	10	0.0074	5.332	0.0265
Error	0.0083	6	0.0014		
Total	0.6634	30			

STATIONARY POINT ANALYSIS

The stationary point is located at:

x(1) = -3.2096
x(2) = 11.3414
x(3) = -0.1812
x(4) = -1.4635

with a probability of cut of:

y = 1.0454

Runway 1 -- Expected Number of Craters to Fill

ANALYSIS OF VARIANCE

Source	Sum of Squares	d.f.	Mean Squares	F	p
Regression	9.2691	14	0.6621	250.291	0.0000
Lack of Fit	1.8598	10	0.1860	70.307	0.0000
Error	0.0159	6	0.0026		
Total	11.1448	30			

STATIONARY POINT ANALYSIS

The stationary point is located at:

x(1) = 16.3333
x(2) = 12.9911
x(3) = -7.7877
x(4) = 10.5375

with an expected number of craters to fill:

y = -2.5010

Runway 2 -- Expected Number of Craters to Fill

ANALYSIS OF VARIANCE

Source	Sum of Squares	d.f.	Mean Squares	F	p
Regression	0.4771	14	0.0341	22.910	0.0005
Lack of Fit	0.0280	10	0.0028	1.886	0.2250
Error	0.0089	6	0.0015		
Total	0.5140	30			

STATIONARY POINT ANALYSIS

The stationary point is located at:

x(1) = 8.5805
x(2) = 20.9471
x(3) = -13.9772
x(4) = -12.3555

with an expected number of craters to fill:

y = 1.9396

Combined Runways -- Expected Number of Craters to Fill

ANALYSIS OF VARIANCE

Source	Sum of Squares	d.f.	Mean Squares	F	p
Regression	0.8259	14	0.0590	38.715	0.0001
Lack of Fit	0.0991	10	0.0099	6.503	0.0163
Error	0.0091	6	0.0015		
Total	0.9342	30			

STATIONARY POINT ANALYSIS

The stationary point is located at:

x(1) = -1.6356
x(2) = 17.0713
x(3) = 0.0215
x(4) = 1.7389

with an expected number of craters to fill:

y = 1.0481

Runway 1 -- Expected Area to Fill

ANALYSIS OF VARIANCE

Source	Sum of Squares	d.f.	Mean Squares	F	p
Regression	1754980	14	125355	242.624	0.0000
Lack of Fit	369613	10	36961	71.538	0.0000
Error	3100	6	517		
Total	2127693	30			

STATIONARY POINT ANALYSIS

The stationary point is located at:

x(1) = -13.2265
x(2) = -12.9150
x(3) = 4.0556
x(4) = -23.8011

with an expected area to fill:

y = 1750.1215

Runway 2 -- Expected Area to Fill

A N A L Y S I S   O F   V A R I A N C E

Source	Sum of Squares	d.f.	Mean Squares	F	p
Regression	127085	14	9077	13.557	0.0021
Lack of Fit	12200	10	1220	1.822	0.2389
Error	4017	6	670		
Total	143303	30			

S T A T I O N A R Y   P O I N T   A N A L Y S I S

The stationary point is located at:

x(1) = -3.8969
x(2) = -6.4726
x(3) = 3.3158
x(4) = 7.6749

with an expected area to fill:

y = 12.8086

Combined Runways -- Expected Area to Fill

A N A L Y S I S   O F   V A R I A N C E

Source	Sum of Squares	d.f.	Mean Squares	F	p
Regression	119780	14	8556	31.987	0.0002
Lack of Fit	18485	10	1849	6.911	0.0140
Error	1605	6	267		
Total	139871	30			

S T A T I O N A R Y   P O I N T   A N A L Y S I S

The stationary point is located at:

x(1) = 0.8837
x(2) = 9.2654
x(3) = -0.6872
x(4) = 4.1461

with an expected area to fill:

y = 214.5446

Mk-84 Results

Combined Runways -- Probability of Cut

ANALYSIS OF VARIANCE

Source	Sum of Squares	d.f.	Mean Squares	F	p
Regression	1.1096	14	0.0793	61.189	0.0000
Lack of Fit	0.0499	10	0.0050	3.855	0.1159
Error	0.0078	6	0.0013		
Total	1.1673	30			

STATIONARY POINT ANALYSIS

The stationary point is located at:

x(1) = 25.2622
x(2) = 2.4511
x(3) = 1.2563
x(4) = 11.8698

with a probability of cut of:

y = -1.9638

APPENDIX K

GLOSSARY

AAA - Anti-aircraft artillery

AAP - Attack Assessment Program

AAPMOD - Attack Assessment Program--MODIFIED

AAPMSN - Used to generate AAPMOD mission input file

AAPTGT - Used to create and maintain AAPMOD target databases

AAPWPN - Used to create and maintain AAPMOD weapon and pattern databases

AFB - Air Force Base

AFIT - Air Force Institute of Technology

aggregated - specific data which is summarized into more general form to reduce input requirements for simulation models. For instance, the number of tanks destroyed might be the value of interest in simulations of a specific battle while tons of munitions might be the indicator of a similar statistic for a theater-level model.

AHAB - Attacking Hardened Air Bases, a RAND simulation model

algorithm - a procedure for solving a mathematical problem

alias - in fractional factorial designs, an alias is a group of interactions which are confounded or confused with a main effect or another interaction. Each member of the alias can affect the system response being studied, but too few data points are used in fractional factorial designs to distinguish among the members of the alias.

alpha - level of significance (1 - confidence level). Probability that the test fails when in actuality it does not.

ANOVA - analysis of variance. A means of comparing statistical entities and performing other tests by comparing the variance of the entities. If the variances of the entities are similar, the entities are assumed to be the same.

APC - Advanced Personal Computer by NEC Information Systems

crater - one effect of munition impacts on a target. The crater is the physical depression which forms when the explosive force of the munition ejects earth or other material from the immediate vicinity of the munition impact point. The term can also include areas of buckling and cracking which extend beyond the physical hole.

deflection - right or left of a target. Deflection is referenced perpendicular to the munition's ground track.

DEP - deflection error probable

deterministic - a problem which can represented with certainty by mathematical equations. This is contrasted with stochastic processes in which particular outcomes may vary when tested several times under the identical circumstances (within the analyst's ability to control the circumstances).

DEV - deviation

discrete - a variable which can assume only fixed values. For instance, one bomb, one sortie, 0.05 probability, etc.

dud - a munition which did not function properly

editor - computer software which facilitates creation of or changes to computer files.

execution - computer processing of a computer program

extrapolate - an estimation of unknown data based on known data

factorial - a statistical design in which all possible combinations of the independent variables are tested. The levels of the variables are usually restricted to either two or three to limit the possible number of combinations.

finite - measurable

fit - the process of calculating a mathematical equation which either contains all design points of interest or minimizes the deviations from the actual points.

FLIR - forward looking infrared. A device used at night or during poor visibility conditions.

floating point coprocessor - an arithmetic processing unit which performs math calculations more quickly than normal microcomputer processors

FORTRAN - Short for "Formula Translator." A high-level computer programming language available on many large and small computers. FORTRAN is designed for rapid scientific calculations.

fractional factorial - a statistical design which uses a portion of the full factorial set of combinations. The number of required design points is reduced, but aliases result.

F-statistic - the test statistic for ANOVA. Used to compare the variance of two or more entities.

fuze - the device which causes a munition to function. In many cases, a fuze is a small explosive device which triggers the larger, but more stable, charge within the main munition.

GST - Graduate Strategic and Tactical Sciences

hardened - potential targets which have been modified to withstand attack.

HQ - headquarters

hyperplane - the surface described by a mathematical equation in more than two independent variables. Conceptually, a plane results when linear combinations of two independent variables are formed. A hyperplane uses the same idea, but the conceptualization is more difficult due to the lack of physical examples.

IBM - International Business Machines

intervalometer - a device which sends release pulses to weapons stations at a selected time interval

intrinsic - a pre-defined computer function which is available for use by the programmer

inverse (exact) - a precise method of generating random variates

iteration - one pass through a particular algorithm. In this analysis, number of samples, replications, and iterations have similar meanings, depending on the context.

JMEM - Joint Munitions Effectiveness Manuals

Lanchester equations - differential equations which purport to describe various types of military conflict

language - a system of computer constructs which the programmer can combine to make desired calculations and print desired output

least-squares - a mathematical method which fits an equation to a set of data points

lethality - the ability of a munition to achieve a desired level of effectiveness

linear - characterized by straight lines. In mathematical terms, linear conditions mean that first-order terms are considered; there are no interaction terms or squared terms.

mainframe - a large computer capable of many operations per second. Mainframe computers are not portable and require extensive support facilities.

MANOVA - multivariate analysis of variance

MCL - minimum clear length. The minimum length required by an aircraft to takeoff and/or land successfully.

MCW - minimum clear width. The minimum width required by an aircraft to takeoff and/or land successfully.

microcomputer - A self-contained computer, relatively less capable and smaller than mainframe computers. In many instances, can be transported easily.

model - a representation of a real-world situation. Models usually employ simplifying assumptions to permit analysis in a timely manner. The assumptions can be the most limiting factor of a particular model.

modulus - an arithmetic operation whereby a remainder is produced. For instance, 11 modulus 3 is 2 since 3 divides into 11 three times with a remainder of 2.

Monte Carlo - a simulation technique which uses random numbers to evaluate a stochastic process. A complex situation may require many such numbers for a single iteration. Many iterations are performed and then averaged to calculate an "expected" response from the system under average circumstances.

MS - mean square; also, Master of Science.

normal - a probability distribution which is characterized by the "bell" curve. The normal distribution is used extensively in modeling various error tolerances and occurrences.

optima - the most desirable or beneficial combination of independent variables giving the best system response.

p-value - the critical level of significance at which a particular test would change answers. For instance, a p-value of 0.05 might indicate that, with a desired level of confidence lower than this value, a certain statistic "passed" the test, while a higher level of confidence would result in "failing" the test. In statistical experiments, the level of confidence is set ahead of time, so the p-value is compared to the preset level of confidence. Suppose that the level of confidence was chosen to be 0.10. A p-value of 0.05 would mean the test had been "passed" (actually, the test failed to reject) while a p-value of 0.15 would "fail" the test.

pair - two bombs which are released simultaneously from weapons stations

PASCAL - a programming language which uses structured programming and emphasizes calculation and mathematical formulations

polygon - a shape which is defined by several planes

polynomial - an equation which does not include complex functional relationships. The independent variables may be combined only with integer exponents. The variables can multiply one another to produce interaction terms.

portable - computer programs which can run on a wide variety of computers. For microcomputers, portability can also include the ability to physically transport the machine from one location to another.

power of a test - the probability of incorrectly failing to reject a null hypothesis (i.e. thinking the test passed when it should not)

predicted response - the result calculated by inserting values for independent variables into a regression equation

processor - that portion of an electronic computer which performs arithmetic and other operations

quadratic - a second-order equation

quantifiable - a variable which can be compared to a number scale

RADAR - radio detection and ranging

RAM - random access memory. The memory a microcomputer uses to store data and computer code

random - occurrences which are not predictable in any fashion

regression - a mathematical procedure by which an equation is calculated to represent a set of data points.

REP - range error probable

replication - a repetition of a particular design point using a different random number stream. Replications are used to determine the variance of a particular system response.

resolution - the degree to which a particular fractional factorial design avoids aliasing. Resolution III designs have no main effects aliased with one another, resolution IV designs have no main effects aliased with each other or 2-factor interactions, and resolution V designs have no main effect or 2-factor interaction aliased with any other main effect or 2-factor interaction. Increasing numbers of design points are required to achieve the higher resolutions.

RSM - response surface methodology

saddle point - a stationary point which is neither a maximum nor a minimum; in 2-dimension space, an inflection point.

SAM - surface-to-air missile

screening - a statistical experiment which is performed to determine the significance of various independent variables

sectioning - a method of response surface methodology which searches for optimal operating conditions. One independent variable is considered at a time. The method of steepest ascent is preferred over sectioning.

seed - the starting number for a random number generator

sensitivity - the degree to which a system response may be affected by changes in independent variables, probability distributions, and assumptions

sigma - the standard deviation of a normal distribution

significance - a measure of the desired degree of accuracy. The level of significance is the probability of incorrectly failing to reject the null hypothesis (i.e. assuming the test passed). Failing to reject near the level of significance results in low power of the test.

singles - the release of one munition at a time

SS - sum of squares

star - the physical representation of a central composite design. In three independent variables, the factorial portion of the design forms a box while the axial points form the star.

stationary point - a point which may be a local minimum, a local maximum, or a saddle point. An optimum point must occur at a stationary point, but not all stationary points will be optimum.

steepest ascent - used in first-order response surface methodology to find optimal operating conditions. Steepest ascent is preferred to sectioning since steepest ascent considers any pertinent variables simultaneously in climbing a local gradient.

stick - a series of weapon impacts

Student's t-statistic - used to compare the average values of statistics

suboptimizing - for fixed levels of overall system response or fixed levels of selected independent variables, the remaining independent variables are optimized. If overall system response is fixed, the independent variables of interest are adjusted to the most favorable (least costly) combination to achieve the response. If some of the independent variables are fixed, the remaining variables are adjusted until the optimum system response is achieved.

survivability - the probability of not being killed during a tactical mission

system response - for this analysis, probability of runway cut, expected number of craters to fill to open a minimum clear strip, and the expected area to fill to open a minimum clear strip

TAC - Tactical Air Command or tactical

TAWM - Tactical Air War Model

theater - a large area of operations such as Europe

triangular distribution - a probability distribution completely described by a mean and two endpoints.

USAF - United States Air Force

USAFE - United Stated Air Forces in Europe

validation - comparison of a model and its output to real-world results

variate - usually a random variate: a variable which behaves in accordance with a particular probability distribution

verification - checks of computer code to ensure that the computer program accurately reflects the intended model.

x-value - the particular value of an independent variable

y-value - the particular value of a dependent variable

Z-value - the random variate corresponding to a particular level of probability.